ESP32 Guardian Component

1

Generated by Doxygen 1.10.0

1 Guardian - ElectionGuard ESP32 Client

1.1 Overview

This project provides the source code and configuration necessary to build an ElectionGuard guardian client for the ESP32 platform. The guardian client is responsible for securely participating in the key generation ceremony and the tallying phase required by the ElectionGuard protocol. This client communicates via MQTT to a central server (Laptop).

1.2 Project Structure

The project is organized into the following directories and files:

- `**main/**`: Contains the main application entry point and performance testing code.
  - `main.c`: The main application logic that initializes the MQTT client and handles communication with the central server.
  - `test_performance.c`: A performance testing module used to evaluate the efficiency of cryptographic operations used during the ElectionGuard key generation ceremony.
- `**/components**`: Contains reusable modules that encapsulate specific functionalities.
  - `**/wolfssl**`: This component overrides the default settings of the WolfSSL third-party library.

- **/adapter**: Handles all communication aspects of the guardian client.
  - Sets up an MQTT client using the configured Wi-Fi connection.
  - Manages incoming and outgoing MQTT messages.
  - Serializes and deserializes data for transmission.
- **/model**: Contains the core business logic and cryptographic operations required by the ElectionGuard protocol.
  - Implements the key ceremony and the distributed decryption
  - Provides an abstraction layer for the underlying cryptographic primitives.

1.3 Functionality

The `main.c` file initializes the ESP32, connects to the configured Wi-Fi network, and starts the MQTT client using the `adapter` component. The `adapter` component then handles communication with the central server, receiving instructions and sending responses. The `model` component implements the cryptographic logic required to perform the ElectionGuard operations.

The `test_performance.c` file provides a means to benchmark the performance of the cryptographic operations used in the `model` component. This is crucial for ensuring that the guardian client can perform its tasks within a reasonable timeframe on the resource-constrained ESP32 platform.

1.4 Building and Flashing

This project is designed to be built using the ESP-IDF framework.

1. **Install ESP-IDF version 5.2:**
2. **Clone the Repository:** Clone this repository to your local machine.
3. **Configure the Project:**
   - Navigate to the project directory in your terminal.
   - Run `idf.py menuconfig` to open the ESP-IDF configuration menu.
   - **Wi-Fi Configuration:** Configure the Wi-Fi SSID and password to connect to your Access Point. This is essential for the guardian to communicate with the MQTT broker.
   - **MQTT Broker Configuration:** Set the MQTT broker address (IP address or hostname) and port. This should match the configuration of the MQTT broker running on the central server.
   - **Other Settings:** Configure any other project-specific settings as needed.
4. **Build the Project:** Run `idf.py build` to compile the project.
5. **Flash the Firmware:** Run `idf.py flash monitor` to flash the firmware to your ESP32 and monitor the serial output.

1.5 Testing and Performance Evaluation

To evaluate the performance of the cryptographic operations, you can use the `test_performance.c` module.

1. **Enable Performance Testing:** Modify `main.c` to call the functions in `test_performance.c`. You need to comment out the MQTT client initialization code to isolate the performance tests.
2. **Build and Flash:** Build and flash the firmware as described above.
3. **Monitor Serial Output:** The serial output will display the results of the performance tests, including the execution time for various cryptographic operations.

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 File Documentation

3.1 D:/Dokuments/IoT-election/guardian/components/adapter/adapter.c File Reference

```
#include "adapter.h"
```

Functions

- void mqtt_event_handler (void ∗handler_args, esp_event_base_t base, int32_t event_id, void ∗event_data)
- void mqtt_app_start (void)

  Start the MQTT client.

Variables

- uint8_t mac [6] = {0}
- int pubkey_count = 0
- int backup_count = 0
- int max_guardians = 0
- int quorum
- ElectionKeyPair guardian
- ElectionKeyPair ∗ pubkey_map
- ElectionPartialKeyPairBackup ∗ backup_map

3.1.1 Function Documentation

mqtt_app_start()

```
void mqtt_app_start (
            void  )
```

Start the MQTT client.

This function initializes the MQTT client, registers the event handler and starts the client.

mqtt_event_handler()

```
void mqtt_event_handler (
            void ∗ handler_args,
            esp_event_base_t base,
            int32_t event_id,
            void ∗ event_data )
```

3.1.2 Variable Documentation

backup_count

```
int backup_count = 0
```

backup_map

```
ElectionPartialKeyPairBackup∗ backup_map
```

guardian

```
ElectionKeyPair guardian
```

mac

```
uint8_t mac[6] = {0}
```

max_guardians

```
int max_guardians = 0
```

pubkey_count

```
int pubkey_count = 0
```

pubkey_map

```
ElectionKeyPair* pubkey_map
```

quorum

```
int quorum
```

3.2 D:/Dokuments/IoT-election/guardian/components/adapter/include/adapter.h File Reference

```
#include "adapter.h"
#include "model.h"
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include "esp_wifi.h"
#include "esp_system.h"
#include "esp_event.h"
#include "esp_netif.h"
#include "protocol_examples_common.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"
#include "freertos/queue.h"
#include "lwip/sockets.h"
#include "lwip/dns.h"
#include "lwip/netdb.h"
#include "esp_log.h"
#include "mqtt_client.h"
#include "esp_mac.h"
#include "serialize.h"
#include "test_decrypt.h"
```

Functions

- void mqtt_app_start (void)

  Start the MQTT client.

3.2.1 Function Documentation

mqtt_app_start()

```
void mqtt_app_start (
            void )
```

Start the MQTT client.

This function initializes the MQTT client, registers the event handler and starts the client.

3.3 adapter.h

Go to the documentation of this file.

```
#ifndef ADAPTER_H
#define ADAPTER_H

#include "adapter.h"
#include "model.h"
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include "esp_wifi.h"
#include "esp_system.h"
#include "esp_event.h"
#include "esp_netif.h"
#include "protocol_examples_common.h"

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"
#include "freertos/queue.h"

#include "lwip/sockets.h"
#include "lwip/dns.h"
#include "lwip/netdb.h"

#include "esp_log.h"
#include "mqtt_client.h"
#include "esp_mac.h"
#include "serialize.h"
#include "test_decrypt.h"


void mqtt_app_start(void);
#endif // ADAPTER_H
```

3.4 D:/Dokuments/IoT-election/guardian/components/adapter/util/include/serialize.h File Reference

```
#include "crypto_utils.h"
#include "buff.pb-c.h"
#include "tally.pb-c.h"
#include <wolfssl/wolfcrypt/wolfmath.h>
```

Functions

- uint8_t ∗ serialize_election_partial_key_verification (ElectionPartialKeyVerification ∗verification, unsigned ∗len)

  Serializes an ElectionPartialKeyVerification object into a byte array.
- uint8_t ∗ serialize_election_partial_key_backup (ElectionPartialKeyPairBackup ∗backup, unsigned ∗len)

  Serializes an ElectionPartialKeyPairBackup object into a byte array.
- uint8_t ∗ serialize_election_key_pair (ElectionKeyPair ∗key_pair, unsigned ∗len)

  Serialize an ElectionKeyPair struct into a byte array.
- uint8_t ∗ serialize_DecryptionShare (DecryptionShare ∗share, unsigned ∗len)

  Serializes a DecryptionShare object into a byte array.
- int deserialize_election_partial_key_verification (uint8_t ∗buffer, unsigned len, ElectionPartialKeyVerification ∗verification)

  Deserializes a byte array into an ElectionPartialKeyVerification struct.
- int deserialize_election_partial_key_backup (uint8_t ∗buffer, unsigned len, ElectionPartialKeyPairBackup ∗backup)

  Deserializes a byte array into an ElectionPartialKeyPairBackup struct.
- int deserialize_election_key_pair (uint8_t ∗buffer, unsigned len, ElectionKeyPair ∗key_pair)

  Deserializes a byte array into an ElectionKeyPair struct.
- int deserialize_ciphertext_tally (uint8_t ∗buffer, unsigned len, CiphertextTally ∗ciphertally)

  Deserializes a byte array into a CiphertextTally struct.

3.4.1 Function Documentation

deserialize_ciphertext_tally()

```
int deserialize_ciphertext_tally (
            uint8_t * buffer,
            unsigned len,
            CiphertextTally * ciphertally )
```

Deserializes a byte array into a CiphertextTally struct.

This function takes a byte array and its length, and deserializes it into a CiphertextTally struct using Protocol Buffers. It unpacks the data and populates the tally object with contest and selection information.

Parameters

buffer	Pointer to the byte array containing the serialized CiphertextTally data.
len	The length of the byte array.
ciphertally	Pointer to the CiphertextTally struct to populate with the deserialized data.

Returns

0 on success, or -1 on failure.

deserialize_election_key_pair()

```
int deserialize_election_key_pair (
            uint8_t * buffer,
            unsigned len,
            ElectionKeyPair * key_pair )
```

Deserializes a byte array into an ElectionKeyPair struct.

This function takes a byte array and its length, and deserializes it into an ElectionKeyPair struct using Protocol Buffers. It unpacks the data, copies the guardian ID, deserializes the public key and the election polynomial.

Parameters

buffer	Pointer to the byte array containing the serialized ElectionKeyPair data.
len	The length of the byte array.
key_pair	Pointer to the ElectionKeyPair struct to populate with the deserialized data.

Returns

0 on success, or 1 on failure.

deserialize_election_partial_key_backup()

```
int deserialize_election_partial_key_backup (
            uint8_t * buffer,
            unsigned len,
            ElectionPartialKeyPairBackup * backup )
```

Deserializes a byte array into an ElectionPartialKeyPairBackup struct.

This function takes a byte array and its length, and deserializes it into an ElectionPartialKeyPairBackup struct using Protocol Buffers. It unpacks the data and copies the sender, receiver, and the encrypted coordinate (HashedElGamalCiphertext).

Parameters

buffer	Pointer to the byte array containing the serialized ElectionPartialKeyPairBackup data.
len	The length of the byte array.
backup	Pointer to the ElectionPartialKeyPairBackup struct to populate with the deserialized data.

Returns

0 on success, or 1 on failure.

deserialize_election_partial_key_verification()

```
int deserialize_election_partial_key_verification (
            uint8_t * buffer,
            unsigned len,
            ElectionPartialKeyVerification * verification )
```

Deserializes a byte array into an ElectionPartialKeyVerification struct.

This function takes a byte array and its length, and deserializes it into an ElectionPartialKeyVerification struct using Protocol Buffers. It unpacks the data and copies the sender, receiver, verifier, and the verification status.

Parameters

buffer	Pointer to the byte array containing the serialized ElectionPartialKeyVerification data.
len	The length of the byte array.
verification	Pointer to the ElectionPartialKeyVerification struct to populate with the deserialized data.

Returns

0 on success, or 1 on failure.

serialize_DecryptionShare()

```
uint8_t * serialize_DecryptionShare (
            DecryptionShare * share,
            unsigned * len )
```

Serializes a DecryptionShare object into a byte array.

This function serializes a DecryptionShare struct into a byte array using Protocol Buffers. It includes information about the guardian, contests, and selections, along with their corresponding decryption proofs.

Parameters

share	Pointer to the DecryptionShare object to serialize.
len	Pointer to an unsigned integer that will store the length of the serialized byte array.

Returns

A pointer to the serialized byte array. The caller is responsible for freeing the allocated memory.

serialize_election_key_pair()

```
uint8_t * serialize_election_key_pair (
            ElectionKeyPair * key_pair,
            unsigned * len )
```

Serialize an ElectionKeyPair struct into a byte array.

This function takes an ElectionKeyPair struct and serializes it into a byte array using Protocol Buffers. The serialized data includes the guardian ID, public key, and the election polynomial with its coefficients and Schnorr proofs.

Parameters

key_pair	Pointer to the ElectionKeyPair object to serialize.
len	Pointer to an unsigned integer that will store the length of the serialized byte array.

Returns

A pointer to the serialized byte array. The caller is responsible for freeing the allocated memory.

serialize_election_partial_key_backup()

```
uint8_t * serialize_election_partial_key_backup (
            ElectionPartialKeyPairBackup * backup,
            unsigned * len )
```

Serializes an ElectionPartialKeyPairBackup object into a byte array.

This function serializes an ElectionPartialKeyPairBackup struct into a byte array using Protocol Buffers. The serialized data includes the sender, receiver, and the encrypted coordinate (HashedElGamalCiphertext).

Parameters

backup	Pointer to the ElectionPartialKeyPairBackup object to serialize.
len	Pointer to an unsigned integer that will store the length of the serialized byte array.

Returns

A pointer to the serialized byte array. The caller is responsible for freeing the allocated memory.

serialize_election_partial_key_verification()

```
uint8_t * serialize_election_partial_key_verification (
            ElectionPartialKeyVerification * verification,
            unsigned * len )
```

Serializes an ElectionPartialKeyVerification object into a byte array.

This function serializes an ElectionPartialKeyVerification struct into a byte array using Protocol Buffers. The serialized data includes the sender, receiver, verifier, and the verification status.

Parameters

verification	Pointer to the ElectionPartialKeyVerification object to serialize.
len	Pointer to an unsigned integer that will store the length of the serialized byte array.

Returns

A pointer to the serialized byte array. The caller is responsible for freeing the allocated memory.

3.5 serialize.h

Go to the documentation of this file.

```
#ifndef SERIALIZE_H
#define SERIALIZE_H
```

```
#include "crypto_utils.h"
#include "buff.pb-c.h"
#include "tally.pb-c.h"
#include <wolfssl/wolfcrypt/wolfmath.h>


uint8_t* serialize_election_partial_key_verification(ElectionPartialKeyVerification* verification,
      unsigned* len);
uint8_t* serialize_election_partial_key_backup(ElectionPartialKeyPairBackup* backup, unsigned* len);
uint8_t* serialize_election_key_pair(ElectionKeyPair* key_pair, unsigned* len);
uint8_t* serialize_DecryptionShare(DecryptionShare* share, unsigned* len);

int deserialize_election_partial_key_verification(uint8_t* buffer, unsigned len,
      ElectionPartialKeyVerification* verification);
int deserialize_election_partial_key_backup(uint8_t* buffer, unsigned len,
      ElectionPartialKeyPairBackup* backup);
int deserialize_election_key_pair(uint8_t* buffer, unsigned len, ElectionKeyPair* key_pair);
int deserialize_ciphertext_tally(uint8_t *buffer, unsigned len, CiphertextTally* ciphertally);

#endif // SERIALIZE_H


```

3.6 D:/Dokuments/IoT-election/guardian/components/adapter/util/include/test_decrypt.h File Reference

```
#include "test_decrypt.h"
#include "model.h"
#include "crypto_utils.h"
#include <stdio.h>
#include "esp_timer.h"
#include <math.h>
#include "esp_task_wdt.h"
```

Functions

- void perform_measurement (ElectionKeyPair ∗guardian, CiphertextTally ∗ciphertally)

  Performs timing measurements for decryption operations and calculates statistics.

3.6.1 Function Documentation

perform_measurement()

```
void perform_measurement (
            ElectionKeyPair * guardian,
            CiphertextTally * tally )
```

Performs timing measurements for decryption operations and calculates statistics.

Parameters

guardian	Pointer to the ElectionKeyPair object used for decryption.
tally	Pointer to the CiphertextTally object to be decrypted.

3.7 test_decrypt.h

Go to the documentation of this file.
```
#ifndef TEST_DECRYPT_H
#define TEST_DECRYPT_H
#include "test_decrypt.h"
#include "model.h"
#include "crypto_utils.h"
#include <stdio.h>
#include "esp_timer.h"
#include <math.h>
#include "esp_task_wdt.h"

void perform_measurement(ElectionKeyPair *guardian, CiphertextTally *ciphertally);

#endif // TEST_DECRYPT_H
```

3.8 D:/Dokuments/IoT-election/guardian/components/adapter/util/serialize.c File Reference

```
#include "serialize.h"
```

Functions

- uint8_t * serialize_election_key_pair (ElectionKeyPair *key_pair, unsigned *len)

  Serialize an ElectionKeyPair struct into a byte array.
- int deserialize_election_key_pair (uint8_t *buffer, unsigned len, ElectionKeyPair *key_pair)

  Deserializes a byte array into an ElectionKeyPair struct.
- uint8_t * serialize_election_partial_key_verification (ElectionPartialKeyVerification *verification, unsigned *len)

  Serializes an ElectionPartialKeyVerification object into a byte array.
- int deserialize_election_partial_key_verification (uint8_t *buffer, unsigned len, ElectionPartialKeyVerification *verification)

  Deserializes a byte array into an ElectionPartialKeyVerification struct.
- uint8_t * serialize_election_partial_key_backup (ElectionPartialKeyPairBackup *backup, unsigned *len)

  Serializes an ElectionPartialKeyPairBackup object into a byte array.
- int deserialize_election_partial_key_backup (uint8_t *buffer, unsigned len, ElectionPartialKeyPairBackup *backup)

  Deserializes a byte array into an ElectionPartialKeyPairBackup struct.
- int deserialize_ciphertext_tally (uint8_t *buffer, unsigned len, CiphertextTally *ciphertally)

  Deserializes a byte array into a CiphertextTally struct.
- uint8_t * serialize_DecryptionShare (DecryptionShare *share, unsigned *len)

  Serializes a DecryptionShare object into a byte array.

3.8.1 Function Documentation

deserialize_ciphertext_tally()

```
int deserialize_ciphertext_tally (
            uint8_t * buffer,
            unsigned len,
            CiphertextTally * ciphertally )
```

Deserializes a byte array into a CiphertextTally struct.

This function takes a byte array and its length, and deserializes it into a CiphertextTally struct using Protocol Buffers. It unpacks the data and populates the tally object with contest and selection information.

Parameters

buffer	Pointer to the byte array containing the serialized CiphertextTally data.
len	The length of the byte array.
ciphertally	Pointer to the CiphertextTally struct to populate with the deserialized data.

Returns

0 on success, or -1 on failure.

deserialize_election_key_pair()

```
int deserialize_election_key_pair (
            uint8_t * buffer,
            unsigned len,
            ElectionKeyPair * key_pair )
```

Deserializes a byte array into an ElectionKeyPair struct.

This function takes a byte array and its length, and deserializes it into an ElectionKeyPair struct using Protocol Buffers. It unpacks the data, copies the guardian ID, deserializes the public key and the election polynomial.

Parameters

buffer	Pointer to the byte array containing the serialized ElectionKeyPair data.
len	The length of the byte array.
key_pair	Pointer to the ElectionKeyPair struct to populate with the deserialized data.

Returns

0 on success, or 1 on failure.

deserialize_election_partial_key_backup()

```
int deserialize_election_partial_key_backup (
            uint8_t * buffer,
            unsigned len,
            ElectionPartialKeyPairBackup * backup )
```

Deserializes a byte array into an ElectionPartialKeyPairBackup struct.

This function takes a byte array and its length, and deserializes it into an ElectionPartialKeyPairBackup struct using Protocol Buffers. It unpacks the data and copies the sender, receiver, and the encrypted coordinate (HashedElGamalCiphertext).

Parameters

buffer	Pointer to the byte array containing the serialized ElectionPartialKeyPairBackup data.
len	The length of the byte array.
backup	Pointer to the ElectionPartialKeyPairBackup struct to populate with the deserialized data.

Returns

0 on success, or 1 on failure.

deserialize_election_partial_key_verification()

```
int deserialize_election_partial_key_verification (
            uint8_t * buffer,
            unsigned len,
            ElectionPartialKeyVerification * verification )
```

Deserializes a byte array into an ElectionPartialKeyVerification struct.

This function takes a byte array and its length, and deserializes it into an ElectionPartialKeyVerification struct using Protocol Buffers. It unpacks the data and copies the sender, receiver, verifier, and the verification status.

Parameters

buffer	Pointer to the byte array containing the serialized ElectionPartialKeyVerification data.
len	The length of the byte array.
verification	Pointer to the ElectionPartialKeyVerification struct to populate with the deserialized data.

Returns

0 on success, or 1 on failure.

serialize_DecryptionShare()

```
uint8_t * serialize_DecryptionShare (
            DecryptionShare * share,
            unsigned * len )
```

Serializes a DecryptionShare object into a byte array.

This function serializes a DecryptionShare struct into a byte array using Protocol Buffers. It includes information about the guardian, contests, and selections, along with their corresponding decryption proofs.

Parameters

share	Pointer to the DecryptionShare object to serialize.
len	Pointer to an unsigned integer that will store the length of the serialized byte array.

Returns

A pointer to the serialized byte array. The caller is responsible for freeing the allocated memory.

serialize_election_key_pair()

```
uint8_t * serialize_election_key_pair (
            ElectionKeyPair * key_pair,
            unsigned * len )
```

Serialize an ElectionKeyPair struct into a byte array.

This function takes an ElectionKeyPair struct and serializes it into a byte array using Protocol Buffers. The serialized data includes the guardian ID, public key, and the election polynomial with its coefficients and Schnorr proofs.

Parameters

key_pair	Pointer to the ElectionKeyPair object to serialize.
len	Pointer to an unsigned integer that will store the length of the serialized byte array.

Returns

A pointer to the serialized byte array. The caller is responsible for freeing the allocated memory.

serialize_election_partial_key_backup()

```
uint8_t * serialize_election_partial_key_backup (
            ElectionPartialKeyPairBackup * backup,
            unsigned * len )
```

Serializes an ElectionPartialKeyPairBackup object into a byte array.

This function serializes an ElectionPartialKeyPairBackup struct into a byte array using Protocol Buffers. The serialized data includes the sender, receiver, and the encrypted coordinate (HashedElGamalCiphertext).

Parameters

backup	Pointer to the ElectionPartialKeyPairBackup object to serialize.
len	Pointer to an unsigned integer that will store the length of the serialized byte array.

Returns

A pointer to the serialized byte array. The caller is responsible for freeing the allocated memory.

serialize_election_partial_key_verification()

```
uint8_t * serialize_election_partial_key_verification (
            ElectionPartialKeyVerification * verification,
            unsigned * len )
```

Serializes an ElectionPartialKeyVerification object into a byte array.

This function serializes an ElectionPartialKeyVerification struct into a byte array using Protocol Buffers. The serialized data includes the sender, receiver, verifier, and the verification status.

Parameters

verification	Pointer to the ElectionPartialKeyVerification object to serialize.
len	Pointer to an unsigned integer that will store the length of the serialized byte array.

Returns

A pointer to the serialized byte array. The caller is responsible for freeing the allocated memory.

3.9 D:/Dokuments/IoT-election/guardian/components/adapter/util/test_decrypt.c File Reference

```
#include "test_decrypt.h"
```

Macros

- #define MEASUREMENT_RUNS 30

Functions

- void perform_measurement (ElectionKeyPair ∗guardian, CiphertextTally ∗tally)

    Performs timing measurements for decryption operations and calculates statistics.

3.9.1 Macro Definition Documentation

MEASUREMENT_RUNS

```
#define MEASUREMENT_RUNS 30
```

3.9.2 Function Documentation

perform_measurement()

```
void perform_measurement (
            ElectionKeyPair * guardian,
            CiphertextTally * tally )
```

Performs timing measurements for decryption operations and calculates statistics.

Parameters

guardian	Pointer to the ElectionKeyPair object used for decryption.
tally	Pointer to the CiphertextTally object to be decrypted.

3.10 D:/Dokuments/IoT-election/guardian/components/model/include/model.h File Reference

```
#include "utils.h"
#include "crypto_utils.h"
#include "model.h"
```

Functions

- void test_hash ()
- int combine_election_public_keys (ElectionKeyPair ∗guardian, ElectionKeyPair ∗pubkey_map, size_t count, ElectionJointKey ∗joint_key)

  Combines individual election public keys into a joint public key.
- int generate_election_key_pair (int quorum, ElectionKeyPair ∗key_pair)

  Generates an election key pair, including the public key, private key, polynomial coefficients, and commitments.
- int generate_election_partial_key_backup (ElectionKeyPair ∗sender, ElectionKeyPair ∗receiver, ElectionPartialKeyPairBackup ∗backup)

  Generates an election partial key backup for sharing between guardians.
- int verify_election_partial_key_backup (ElectionKeyPair ∗receiver, ElectionKeyPair ∗sender, ElectionPartialKeyPairBackup ∗backup, ElectionPartialKeyVerification ∗verification)

  Verifies an election partial key backup to confirm it contains a point on the owner's polynomial.
- int compute_decryption_share (ElectionKeyPair ∗guardian, CiphertextTally ∗ciphertally, DecryptionShare ∗share)

  Computes a decryption share for a given ciphertext tally using a guardian's private key.

3.10.1 Function Documentation

combine_election_public_keys()

```
int combine_election_public_keys (
            ElectionKeyPair * guardian,
            ElectionKeyPair * pubkey_map,
            size_t count,
            ElectionJointKey * joint_key )
```

Combines individual election public keys into a joint public key.

This function aggregates the public keys of multiple guardians to form a joint public key, which happens at the end of the key ceremony. It also generates a commitment hash of the combined public key.

Parameters

guardian	An `ElectionKeyPair` representing the current guardian.
pubkey_map	An array of `ElectionKeyPair` structures, each containing a guardian's public key.
count	The number of guardians (and thus the number of public keys in `pubkey_map`).
joint_key	A pointer to an `ElectionJointKey` struct where the resulting joint public key and commitment hash will be stored. The caller must allocate memory for this struct before calling the function.

Returns

0 on success.

Note

The function computes both the joint public key and its corresponding commitment hash.

compute_decryption_share()

```
int compute_decryption_share (
            ElectionKeyPair * guardian,
            CiphertextTally * ciphertally,
            DecryptionShare * share )
```

Computes a decryption share for a given ciphertext tally using a guardian's private key.

Each guardian computes a decryption share for each contest in the ciphertext tally. These shares are later combined to decrypt the tally and reveal the election results.

Parameters

guardian	The `ElectionKeyPair` of the guardian computing the decryption share.
ciphertally	A pointer to the `CiphertextTally` struct containing the encrypted tallies for each contest.
share	A pointer to a `DecryptionShare` struct where the computed decryption share will be stored. The caller must allocate memory for this struct before calling the function.

Returns

0 on success.

Note

The function iterates through each contest in the ciphertext tally and computes a decryption share for it.

generate_election_key_pair()

```
int generate_election_key_pair (
            int quorum,
            ElectionKeyPair * key_pair )
```

Generates an election key pair, including the public key, private key, polynomial coefficients, and commitments.

This function generates an election key pair based on a specified quorum. It involves creating a polynomial with a degree determined by the quorum size. The coefficients of this polynomial form the basis for the private key, while commitments to these coefficients constitute the public key.

Parameters

quorum	The number of guardians required to decrypt the election. This determines the degree of the polynomial.
key_pair	A pointer to an `ElectionKeyPair` struct where the generated key pair, polynomial, and related data will be stored. The caller is responsible for allocating the `ElectionKeyPair` structure before calling this function.

Returns

0 on success.

-1 on failure, typically due to memory allocation issues.

Note

This function allocates memory for the public key, private key, and polynomial coefficients. It is crucial to free this memory after use to prevent memory leaks.

```
ElectionKeyPair key_pair;
int quorum = 10;
if (generate_election_key_pair(quorum, &key_pair) == 0) {
  // Use the generated key pair
  // ...
  // Free the allocated memory
  free_ElectionKeyPair(&key_pair);
} else {
  // Handle error
}
```

generate_election_partial_key_backup()

```
int generate_election_partial_key_backup (
               ElectionKeyPair * sender,
               ElectionKeyPair * receiver,
               ElectionPartialKeyPairBackup * backup )
```

Generates an election partial key backup for sharing between guardians.

This function creates a backup of a guardian's partial key, encrypts it using the recipient's public key, and prepares it for secure sharing. The backup includes the encrypted coordinate, sender and receiver identifiers.

Parameters

sender	The `ElectionKeyPair` of the guardian sending the backup.
receiver	The `ElectionKeyPair` of the guardian receiving the backup.
backup	A pointer to an `ElectionPartialKeyPairBackup` struct where the encrypted backup will be stored. The caller must allocate memory for this struct before calling the function.

Returns

0 on success.

-1 on failure.

Note

The function uses the receiver's public key to encrypt a coordinate derived from the sender's polynomial. It's essential to ensure the receiver's public key is valid and trusted.

test_hash()

```
void test_hash ( )
```

verify_election_partial_key_backup()

```
int verify_election_partial_key_backup (
            ElectionKeyPair * receiver,
            ElectionKeyPair * sender,
            ElectionPartialKeyPairBackup * backup,
            ElectionPartialKeyVerification * verification )
```

Verifies an election partial key backup to confirm it contains a point on the owner's polynomial.

This function decrypts the encrypted coordinate from the backup using the receiver's private key and then verifies that the decrypted coordinate corresponds to a point on the sender's polynomial. This ensures the backup is valid and originates from the claimed sender.

Parameters

receiver	The `ElectionKeyPair` of the guardian receiving and verifying the backup.
sender	The `ElectionKeyPair` of the guardian who sent the backup.
backup	A pointer to the `ElectionPartialKeyPairBackup` struct containing the encrypted backup to be verified.
verification	A pointer to an `ElectionPartialKeyVerification` struct where the verification result will be stored. The caller must allocate memory for this struct before calling the function.

Returns

0 on success.

Note

The function updates the `verified` field in the `ElectionPartialKeyVerification` struct to indicate whether the backup is valid.

3.11 model.h

Go to the documentation of this file.

```
#ifndef MODEL_H
#define MODEL_H


#include "utils.h"
#include "crypto_utils.h"
#include "model.h"

void test_hash();
```

```
int combine_election_public_keys(ElectionKeyPair *guardian, ElectionKeyPair *pubkey_map, size_t count,
ElectionJointKey *joint_key);
int generate_election_key_pair(int quorum, ElectionKeyPair *key_pair);
int generate_election_partial_key_backup(ElectionKeyPair *sender, ElectionKeyPair *receiver,
ElectionPartialKeyPairBackup *backup);
int verify_election_partial_key_backup(ElectionKeyPair *receiver, ElectionKeyPair *sender,
ElectionPartialKeyPairBackup *backup, ElectionPartialKeyVerification *verification);
int compute_decryption_share(ElectionKeyPair *guardian, CiphertextTally *ciphertally, DecryptionShare
*share);
#endif // MOD_MATH_H
```

3.12 D:/Dokuments/IoT-election/guardian/components/model/model.c File Reference

```
#include "model.h"
```

Functions

- int generate_election_key_pair (int quorum, ElectionKeyPair *key_pair)
  Generates an election key pair, including the public key, private key, polynomial coefficients, and commitments.
- int generate_election_partial_key_backup (ElectionKeyPair *sender, ElectionKeyPair *receiver, ElectionPartialKeyPairBackup *backup)
  Generates an election partial key backup for sharing between guardians.
- int verify_election_partial_key_backup (ElectionKeyPair *receiver, ElectionKeyPair *sender, ElectionPartialKeyPairBackup *backup, ElectionPartialKeyVerification *verification)
  Verifies an election partial key backup to confirm it contains a point on the owner's polynomial.
- int combine_election_public_keys (ElectionKeyPair *guardian, ElectionKeyPair *pubkey_map, size_t count, ElectionJointKey *joint_key)
  Combines individual election public keys into a joint public key.
- int compute_decryption_share (ElectionKeyPair *guardian, CiphertextTally *ciphertally, DecryptionShare *share)
  Computes a decryption share for a given ciphertext tally using a guardian's private key.

3.12.1 Function Documentation

combine_election_public_keys()

```
int combine_election_public_keys (
            ElectionKeyPair * guardian,
            ElectionKeyPair * pubkey_map,
            size_t count,
            ElectionJointKey * joint_key )
```

Combines individual election public keys into a joint public key.

This function aggregates the public keys of multiple guardians to form a joint public key, which happens at the end of the key ceremony. It also generates a commitment hash of the combined public key.

Parameters

guardian	An `ElectionKeyPair` representing the current guardian.
pubkey_map	An array of `ElectionKeyPair` structures, each containing a guardian's public key.
count	The number of guardians (and thus the number of public keys in `pubkey_map`).
joint_key	A pointer to an `ElectionJointKey` struct where the resulting joint public key and commitment hash will be stored. The caller must allocate memory for this struct before calling the function.

Returns

0 on success.

Note

The function computes both the joint public key and its corresponding commitment hash.

compute_decryption_share()

```
int compute_decryption_share (
            ElectionKeyPair * guardian,
            CiphertextTally * ciphertally,
            DecryptionShare * share )
```

Computes a decryption share for a given ciphertext tally using a guardian's private key.

Each guardian computes a decryption share for each contest in the ciphertext tally. These shares are later combined to decrypt the tally and reveal the election results.

Parameters

guardian	The `ElectionKeyPair` of the guardian computing the decryption share.
ciphertally	A pointer to the `CiphertextTally` struct containing the encrypted tallies for each contest.
share	A pointer to a `DecryptionShare` struct where the computed decryption share will be stored. The caller must allocate memory for this struct before calling the function.

Returns

0 on success.

Note

The function iterates through each contest in the ciphertext tally and computes a decryption share for it.

generate_election_key_pair()

```
int generate_election_key_pair (
            int quorum,
            ElectionKeyPair * key_pair )
```

Generates an election key pair, including the public key, private key, polynomial coefficients, and commitments.

This function generates an election key pair based on a specified quorum. It involves creating a polynomial with a degree determined by the quorum size. The coefficients of this polynomial form the basis for the private key, while commitments to these coefficients constitute the public key.

Parameters

quorum	The number of guardians required to decrypt the election. This determines the degree of the polynomial.
key_pair	A pointer to an ElectionKeyPair struct where the generated key pair, polynomial, and related data will be stored. The caller is responsible for allocating the ElectionKeyPair structure before calling this function.

Returns

0 on success.

-1 on failure, typically due to memory allocation issues.

Note

This function allocates memory for the public key, private key, and polynomial coefficients. It is crucial to free this memory after use to prevent memory leaks.

```
ElectionKeyPair key_pair;
int quorum = 10;
if (generate_election_key_pair(quorum, &key_pair) == 0) {
  // Use the generated key pair
  // ...
  // Free the allocated memory
  free_ElectionKeyPair(&key_pair);
} else {
  // Handle error
}
```

generate_election_partial_key_backup()

```
int generate_election_partial_key_backup (
            ElectionKeyPair * sender,
            ElectionKeyPair * receiver,
            ElectionPartialKeyPairBackup * backup )
```

Generates an election partial key backup for sharing between guardians.

This function creates a backup of a guardian's partial key, encrypts it using the recipient's public key, and prepares it for secure sharing. The backup includes the encrypted coordinate, sender and receiver identifiers.

Parameters

sender	The ElectionKeyPair of the guardian sending the backup.
receiver	The ElectionKeyPair of the guardian receiving the backup.
backup	A pointer to an ElectionPartialKeyPairBackup struct where the encrypted backup will be stored. The caller must allocate memory for this struct before calling the function.

Returns

0 on success.

-1 on failure.

Note

The function uses the receiver's public key to encrypt a coordinate derived from the sender's polynomial. It's essential to ensure the receiver's public key is valid and trusted.

verify_election_partial_key_backup()

```
int verify_election_partial_key_backup (
            ElectionKeyPair * receiver,
            ElectionKeyPair * sender,
            ElectionPartialKeyPairBackup * backup,
            ElectionPartialKeyVerification * verification )
```

Verifies an election partial key backup to confirm it contains a point on the owner's polynomial.

This function decrypts the encrypted coordinate from the backup using the receiver's private key and then verifies that the decrypted coordinate corresponds to a point on the sender's polynomial. This ensures the backup is valid and originates from the claimed sender.

Parameters

receiver	The `ElectionKeyPair` of the guardian receiving and verifying the backup.
sender	The `ElectionKeyPair` of the guardian who sent the backup.
backup	A pointer to the `ElectionPartialKeyPairBackup` struct containing the encrypted backup to be verified.
verification	A pointer to an `ElectionPartialKeyVerification` struct where the verification result will be stored. The caller must allocate memory for this struct before calling the function.

Returns

0 on success.

Note

The function updates the `verified` field in the `ElectionPartialKeyVerification` struct to indicate whether the backup is valid.

3.13 D:/Dokuments/IoT-election/guardian/components/model/util/constants.c File Reference

```
#include "constants.h"
```

Variables

- const unsigned char p_3072 [ ]
- const unsigned char g_3072 [ ]
- const unsigned char q_256 [ ]

3.13.1 Variable Documentation

g_3072

```
const unsigned char g_3072[]
```

p_3072

```
const unsigned char p_3072[]
```

q_256

```
const unsigned char q_256[]
```

Initial value:

```
= {
    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
}
```

3.14 D:/Dokuments/IoT-election/guardian/components/model/util/crypto_utils.c File Reference

```
#include "crypto_utils.h"
```

Functions

- void print_sp_int (sp_int ∗num)

  Prints an sp_int to the ESP_LOGI.
- void print_byte_array (const byte ∗array, int size)

  Prints a byte array to the ESP_LOGI.
- int rand_q (sp_int ∗result)

  Generate a random number below constant Prime Q (small prime)
- int hash (sp_int ∗a, sp_int ∗b, sp_int ∗result)

  Given two mp_ints, calculate their cryptographic hash using SHA256.
- int compute_polynomial_coordinate (uint8_t ∗exponent_modifier, ElectionPolynomial ∗polynomial, sp_int ∗coordinate)

  Computes a single coordinate value of the election polynomial used for sharing.
- int verify_polynomial_coordinate (uint8_t ∗exponent_modifier, ElectionPolynomial ∗polynomial, sp_int ∗coordinate)

  Verifies that a coordinate lies on the election polynomial.
- int hashed_elgamal_encrypt (sp_int ∗message, sp_int ∗nonce, sp_int ∗public_key, sp_int ∗encryption_seed, HashedElGamalCiphertext ∗encrypted_message)

  Encrypts a message using Hashed ElGamal encryption.
- int hashed_elgamal_decrypt (HashedElGamalCiphertext ∗encrypted_message, sp_int ∗secret_key, sp_int ∗encryption_seed, sp_int ∗message)
- int generate_polynomial (ElectionPolynomial ∗polynomial)

Generates a polynomial for sharing election keys using Shamir's Secret Sharing.
- int elgamal_combine_public_keys (ElectionKeyPair ∗guardian, ElectionKeyPair ∗pubkey_map, size_t count, sp_int ∗key)

  Combines multiple ElGamal public keys into a single aggregate key.
- int hash_keys (ElectionKeyPair ∗guardian, ElectionKeyPair ∗pubkey_map, size_t count, sp_int ∗commitment)

  Computes a SHA256 hash of the commitments from multiple guardians.
- int make_chaum_pedersen (sp_int ∗alpha, sp_int ∗beta, sp_int ∗secret, sp_int ∗m, sp_int ∗seed, sp_int ∗hash_header, ChaumPedersenProof ∗proof)

  Generates a Chaum-Pedersen proof to demonstrate that a ciphertext corresponds to a specific plaintext.
- int verify_chaum_pedersen (sp_int ∗public_key, ChaumPedersenProof ∗proof, sp_int ∗alpha, sp_int ∗m)

  Verifies a Chaum-Pedersen proof.
- int compute_decryption_share_for_contest (ElectionKeyPair ∗guardian, CiphertextTallyContest ∗contest, sp_int ∗base_hash, CiphertextDecryptionContest ∗dec_contest)

  Computes a decryption share for a contest.

3.14.1 Function Documentation

compute_decryption_share_for_contest()

```
int compute_decryption_share_for_contest (
            ElectionKeyPair ∗ guardian,
            CiphertextTallyContest ∗ contest,
            sp_int ∗ base_hash,
            CiphertextDecryptionContest ∗ dec_contest )
```

Computes a decryption share for a contest.

This function computes a decryption share for a given contest based on the guardian's key pair and the contest data. It allocates memory for the decryption contest and its selections, copies relevant data, and computes the decryption share for each selection in the contest.

Parameters

guardian	A pointer to the `ElectionKeyPair` structure containing the guardian's key pair.
contest	A pointer to the `CiphertextTallyContest` structure representing the contest data.
base_hash	A pointer to the base hash (`sp_int`) used in the decryption process.
dec_contest	A pointer to the `CiphertextDecryptionContest` structure where the computed decryption share will be stored.

Returns

0 on success, -1 on memory allocation failure.

compute_polynomial_coordinate()

```
int compute_polynomial_coordinate (
            uint8_t ∗ exponent_modifier,
            ElectionPolynomial ∗ polynomial,
            sp_int ∗ coordinate )
```

Computes a single coordinate value of the election polynomial used for sharing.

Parameters

exponent_modifier	Unique modifier (guardian id) for exponent [0, Q]
polynomial	Election polynomial
coordinate	The computed coordinate

Returns

0 on success, -1 on failure

elgamal_combine_public_keys()

```
int elgamal_combine_public_keys (
            ElectionKeyPair * guardian,
            ElectionKeyPair * pubkey_map,
            size_t count,
            sp_int * key )
```

Combines multiple ElGamal public keys into a single aggregate key.

This function combines the public keys of a guardian and a set of other guardians into a single aggregate public key. The aggregate key is computed by multiplying all the individual public keys together modulo a large prime.

Parameters

guardian	A pointer to the `ElectionKeyPair` struct of the primary guardian. Its public key will be included in the combination.
pubkey_map	An array of `ElectionKeyPair` structs representing the other guardians whose public keys will be combined.
count	The number of elements in the `pubkey_map` array.
key	A pointer to an `sp_int` where the resulting combined public key will be stored. The caller must allocate memory for this `sp_int` before calling the function.

Returns

0 on success.

-1 on failure (e.g., if any of the multiplication operations fail).

Note

The function assumes that all public keys are valid ElGamal public keys with respect to the same large prime p.

generate_polynomial()

```
int generate_polynomial (
            ElectionPolynomial * polynomial )
```

Generates a polynomial for sharing election keys using Shamir's Secret Sharing.

This function generates a polynomial of a specified degree, where each coefficient is a secret value. The polynomial is used to share an election key among multiple guardians using Shamir's Secret Sharing scheme. Each guardian receives a share of the secret key, which is an evaluation of the polynomial at a specific point. Any subset of guardians above a threshold can reconstruct the original secret key.

For each coefficient of the polynomial, the function generates a random value, computes a commitment to that value, and creates a Schnorr proof of knowledge for the coefficient.

Parameters

polynomial	A pointer to the `ElectionPolynomial` struct where the generated polynomial will be stored. The caller must allocate memory for the `ElectionPolynomial` struct and set the `num_coefficients` field before calling this function. This function will allocate memory for the `coefficients` array within the `ElectionPolynomial` struct, as well as for the `value`, `commitment`, and `proof` fields within each `PolynomialCoefficient` struct.

Returns

0 on success.

-1 on failure (e.g., memory allocation error).

Note

The function allocates memory for the fields within the `polynomial` structure. It is the caller's responsibility to free this memory when the polynomial is no longer needed to prevent memory leaks. See `ElectionPolynomial` and `PolynomialCoefficient` documentation for details.

hash()

```
int hash (
            sp_int * a,
            sp_int * b,
            sp_int * result )
```

Given two mp_ints, calculate their cryptographic hash using SHA256.

Each value is converted to a hexadecimal string and hashed with a delimiter in between (e.g. H(|a|b|)).

Parameters

a	First element
b	Second element
result	The result of the hash

Returns

0 on success, -1 on failure

hash_keys()

```
int hash_keys (
            ElectionKeyPair * guardian,
            ElectionKeyPair * pubkey_map,
            size_t count,
            sp_int * commitment )
```

Computes a SHA256 hash of the commitments from multiple guardians.

This function calculates a SHA256 hash of the commitments made by a primary guardian and a set of other guardians. The hash serves as a binding value that commits all guardians to their respective commitments.

Parameters

guardian	A pointer to the `ElectionKeyPair` struct of the primary guardian. The commitments from its polynomial will be included in the hash.
pubkey_map	An array of `ElectionKeyPair` structs representing the other guardians whose commitments will be included in the hash.
count	The number of elements in the `pubkey_map` array.
commitment	A pointer to an `sp_int` where the resulting hash value will be stored. The caller must allocate memory for this `sp_int` before calling the function.

Returns

0 on success.

-1 on failure (e.g., if SHA256 initialization or update fails, or memory allocation error).

Note

The function concatenates the commitments of all guardians, separated by delimiters, before hashing the result.

hashed_elgamal_decrypt()

```
int hashed_elgamal_decrypt (
            HashedElGamalCiphertext * encrypted_message,
            sp_int * secret_key,
            sp_int * encryption_seed,
            sp_int * message )
```

Decrypt an ElGamal ciphertext using a known ElGamal secret key

Parameters

encrypted_message	struct containing pad, data, and mac
secret_key	The corresponding ElGamal secret key.
encryption_seed	Encryption seed (Q) for election.
message	Decrypted plaintext message.

Returns

0 on success, -1 on failure

hashed_elgamal_encrypt()

```
int hashed_elgamal_encrypt (
            sp_int * message,
            sp_int * nonce,
            sp_int * public_key,
            sp_int * encryption_seed,
            HashedElGamalCiphertext * encrypted_message )
```

Encrypts a message using Hashed ElGamal encryption.

This function encrypts a given message using the Hashed ElGamal encryption scheme. It generates a ciphertext consisting of a pad, data, and MAC (Message Authentication Code).

Parameters

message	The message (sp_int) to be encrypted.
nonce	A random nonce (sp_int) used for encryption.
public_key	The recipient's public key (sp_int).
encryption_seed	An encryption seed (sp_int) used in the key derivation function.
encrypted_message	A pointer to the HashedElGamalCiphertext structure where the resulting ciphertext will be stored.

Returns

0 on success, 1 on memory allocation failure.

make_chaum_pedersen()

```
int make_chaum_pedersen (
            sp_int * alpha,
            sp_int * beta,
            sp_int * secret,
            sp_int * m,
            sp_int * seed,
            sp_int * hash_header,
            ChaumPedersenProof * proof )
```

Generates a Chaum-Pedersen proof to demonstrate that a ciphertext corresponds to a specific plaintext.

This function generates a Chaum-Pedersen proof, which is a zero-knowledge proof that demonstrates that a given ElGamal ciphertext (alpha, beta) encrypts a specific plaintext value (m). The proof shows that the prover knows the secret (nonce) used to create the ciphertext, without revealing the secret itself.

Parameters

alpha	A pointer to the ElGamal ciphertext's alpha component ((g^r \mod p)), which is a large integer.
beta	A pointer to the ElGamal ciphertext's beta component ((m \cdot y^r \mod p)), which is a large integer.
secret	A pointer to the nonce ((r)) used to encrypt the message, which is a large integer.
m	A pointer to the plaintext message ((m)), which is a large integer.
seed	A pointer to a seed value used to generate random values within the proof.

Parameters

hash_header	A pointer to a header value used when generating the challenge, typically the election extended base hash.
proof	A pointer to the `ChaumPedersenProof` struct where the generated proof will be stored. The caller is responsible for allocating memory for the `ChaumPedersenProof` struct. The function will allocate memory for the `pad`, `data`, `challenge`, and `response` fields within the `ChaumPedersenProof` struct.

Returns

0 on success.

-1 on failure (e.g., memory allocation error).

Note

The function allocates memory for the fields within the `proof` structure. It is the caller's responsibility to free this memory when the proof is no longer needed to prevent memory leaks. See `ChaumPedersenProof` documentation for details.

print_byte_array()

```
void print_byte_array (
             const byte * array,
             int size )
```

Prints a byte array to the ESP_LOGI.

This function converts a byte array to a hexadecimal string and prints it using ESP_LOGI.

Parameters

array	The byte array to print.
size	The size of the byte array.

print_sp_int()

```
void print_sp_int (
             sp_int * num )
```

Prints an sp_int to the ESP_LOGI.

This function converts an sp_int to a hexadecimal string and prints it using ESP_LOGI.

Parameters

num	The sp_int to print.

rand_q()

```
int rand_q (
            sp_int * result )
```

Generate a random number below constant Prime Q (small prime)

Parameters

result	The random number

Returns

0 on success, -1 on failure

verify_chaum_pedersen()

```
int verify_chaum_pedersen (
            sp_int * public_key,
            ChaumPedersenProof * proof,
            sp_int * alpha,
            sp_int * m )
```

Verifies a Chaum-Pedersen proof.

This function verifies that a given Chaum-Pedersen proof is valid with respect to the provided public key

Parameters

public_key	The public key (`sp_int`) used in the Chaum-Pedersen proof.
proof	A pointer to the `ChaumPedersenProof` structure containing the proof data.
alpha	An `sp_int` representing the generator `g` of the group.
m	An `sp_int` representing the message.

Returns

0 if the proof is valid, otherwise logs an error message and returns a non-zero value.

verify_polynomial_coordinate()

```
int verify_polynomial_coordinate (
            uint8_t * exponent_modifier,
            ElectionPolynomial * polynomial,
            sp_int * coordinate )
```

Verifies that a coordinate lies on the election polynomial.

Given a guardian's identifier, a polynomial, and a coordinate, this function verifies that the coordinate corresponds to a point on the polynomial. This is used to validate partial key backups.

Parameters

exponent_modifier	The unique identifier of the guardian (usually sequence order).
polynomial	The election polynomial to verify against.
coordinate	The coordinate to verify.

Returns

1 if the coordinate is on the polynomial, 0 otherwise. Returns -1 on failure.

3.15 D:/Dokuments/IoT-election/guardian/components/model/util/include/constants.h File Reference

Variables

- const unsigned char p_3072 [384]
- const unsigned char g_3072 [384]
- const unsigned char q_256 [32]

3.15.1 Variable Documentation

g_3072

```
const unsigned char g_3072[384]  [extern]
```

p_3072

```
const unsigned char p_3072[384]  [extern]
```

q_256

```
const unsigned char q_256[32]  [extern]
```

3.16 constants.h

Go to the documentation of this file.

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

// The constants for mathematical functions during the election

extern const unsigned char p_3072[384];
extern const unsigned char g_3072[384];
extern const unsigned char q_256[32];

#endif // CONSTANTS_H
```

3.17 D:/Dokuments/IoT-election/guardian/components/model/util/include/crypto_utils.h File Reference

```
#include <wolfssl/wolfcrypt/port/Espressif/esp32-crypt.h>
#include <wolfssl/wolfcrypt/wolfmath.h>
#include <wolfssl/ssl.h>
#include "esp_log.h"
#include "esp_random.h"
#include "constants.h"
```

Data Structures

- struct SchnorrProof
- struct Coefficient
- struct ElectionPolynomial
- struct ElectionKeyPair
- struct HashedElGamalCiphertext
- struct ElectionPartialKeyPairBackup
- struct ElectionPartialKeyVerification
- struct ElectionJointKey
- struct CiphertextTallySelection
- struct CiphertextTallyContest
- struct CiphertextTally
- struct ChaumPedersenProof
- struct CiphertextDecryptionSelection
- struct CiphertextDecryptionContest
- struct DecryptionShare

Macros

- #define BLOCK_SIZE 32
- #define MODIFIED_CHUNK_SIZE (BLOCK_SIZE + 8)

Functions

- int elgamal_combine_public_keys (ElectionKeyPair ∗guardian, ElectionKeyPair ∗pubkey_map, size_t count, sp_int ∗key)

  Combines multiple ElGamal public keys into a single aggregate key.
- int compute_decryption_share_for_contest (ElectionKeyPair ∗guardian, CiphertextTallyContest ∗contest, sp_int ∗base_hash, CiphertextDecryptionContest ∗dec_contest)

  Computes a decryption share for a contest.
- int compute_polynomial_coordinate (uint8_t ∗exponent_modifier, ElectionPolynomial ∗polynomial, sp_int ∗coordinate)

  Computes a single coordinate value of the election polynomial used for sharing.
- int verify_polynomial_coordinate (uint8_t ∗exponent_modifier, ElectionPolynomial ∗polynomial, sp_int ∗coordinate)

  Verifies that a coordinate lies on the election polynomial.
- int hashed_elgamal_encrypt (sp_int ∗message, sp_int ∗nonce, sp_int ∗public_key, sp_int ∗encryption_seed, HashedElGamalCiphertext ∗encrypted_message)

  Encrypts a message using Hashed ElGamal encryption.

- int hashed_elgamal_decrypt (HashedElGamalCiphertext ∗encrypted_message, sp_int ∗secret_key, sp_int ∗encryption_seed, sp_int ∗message)
- int generate_polynomial (ElectionPolynomial ∗polynomial)

  Generates a polynomial for sharing election keys using Shamir's Secret Sharing.
- int hash (sp_int ∗a, sp_int ∗b, sp_int ∗result)

  Given two mp_ints, calculate their cryptographic hash using SHA256.
- int hash_keys (ElectionKeyPair ∗guardian, ElectionKeyPair ∗pubkey_map, size_t count, sp_int ∗commitment)

  Computes a SHA256 hash of the commitments from multiple guardians.
- int rand_q (sp_int ∗result)

  Generate a random number below constant Prime Q (small prime)
- int make_chaum_pedersen (sp_int ∗alpha, sp_int ∗beta, sp_int ∗secret, sp_int ∗m, sp_int ∗seed, sp_int ∗hash_header, ChaumPedersenProof ∗proof)

  Generates a Chaum-Pedersen proof to demonstrate that a ciphertext corresponds to a specific plaintext.
- void print_sp_int (sp_int ∗num)

  Prints an sp_int to the ESP_LOGI.
- void print_byte_array (const byte ∗array, int size)

  Prints a byte array to the ESP_LOGI.

3.17.1 Data Structure Documentation

struct SchnorrProof

Data Fields

sp_int ∗	challenge	
sp_int ∗	commitment	
sp_int ∗	pubkey	
sp_int ∗	response	

struct Coefficient

Data Fields

sp_int ∗	commitment	
SchnorrProof	proof	
sp_int ∗	value	

struct ElectionPolynomial

Data Fields

Coefficient ∗	coefficients	
int	num_coefficients	

struct ElectionKeyPair

Data Fields

uint8_t	guardian_id[6]	
ElectionPolynomial	polynomial	
sp_int ∗	private_key	
sp_int ∗	public_key	

struct HashedElGamalCiphertext

Data Fields

sp_int ∗	data	
sp_int ∗	mac	
sp_int ∗	pad	

struct ElectionPartialKeyPairBackup

Data Fields

HashedElGamalCiphertext	encrypted_coordinate	
uint8_t	receiver[6]	
uint8_t	sender[6]	

struct ElectionPartialKeyVerification

Data Fields

uint8_t	receiver[6]	
uint8_t	sender[6]	
bool	verified	
uint8_t	verifier[6]	

struct ElectionJointKey

Data Fields

sp_int ∗	commitment_hash	
sp_int ∗	joint_key	

struct CiphertextTallySelection

Data Fields

sp_int ∗	ciphertext_data	
sp_int ∗	ciphertext_pad	
char ∗	object_id	

struct CiphertextTallyContest

Data Fields

sp_int ∗	description_hash	
int	num_selections	
char ∗	object_id	
CiphertextTallySelection ∗	selections	
int	sequence_order	

struct CiphertextTally

Data Fields

sp_int ∗	base_hash	
CiphertextTallyContest ∗	contests	
int	num_contest	
char ∗	object_id	

struct ChaumPedersenProof

Data Fields

sp_int ∗	challenge	
sp_int ∗	data	
sp_int ∗	pad	
sp_int ∗	response	

struct CiphertextDecryptionSelection

Data Fields

sp_int ∗	decryption	
uint8_t	guardian_id[6]	
char ∗	object_id	
ChaumPedersenProof	proof	

struct CiphertextDecryptionContest

Data Fields

sp_int ∗	description_hash	
uint8_t	guardian_id[6]	
int	num_selections	
char ∗	object_id	
sp_int ∗	public_key	
CiphertextDecryptionSelection ∗	selections	

struct DecryptionShare

Data Fields

CiphertextDecryptionContest ∗	contests	
uint8_t	guardian_id[6]	
int	num_contest	
char ∗	object_id	
sp_int ∗	public_key	

3.17.2 Macro Definition Documentation

BLOCK_SIZE

```
#define BLOCK_SIZE 32
```

MODIFIED_CHUNK_SIZE

```
#define MODIFIED_CHUNK_SIZE (BLOCK_SIZE + 8)
```

3.17.3 Function Documentation

compute_decryption_share_for_contest()

```
int compute_decryption_share_for_contest (
            ElectionKeyPair * guardian,
            CiphertextTallyContest * contest,
            sp_int * base_hash,
            CiphertextDecryptionContest * dec_contest )
```

Computes a decryption share for a contest.

This function computes a decryption share for a given contest based on the guardian's key pair and the contest data. It allocates memory for the decryption contest and its selections, copies relevant data, and computes the decryption share for each selection in the contest.

Parameters

guardian	A pointer to the `ElectionKeyPair` structure containing the guardian's key pair.
contest	A pointer to the `CiphertextTallyContest` structure representing the contest data.
base_hash	A pointer to the base hash (`sp_int`) used in the decryption process.
dec_contest	A pointer to the `CiphertextDecryptionContest` structure where the computed decryption share will be stored.

Returns

0 on success, -1 on memory allocation failure.

compute_polynomial_coordinate()

```
int compute_polynomial_coordinate (
            uint8_t * exponent_modifier,
            ElectionPolynomial * polynomial,
            sp_int * coordinate )
```

Computes a single coordinate value of the election polynomial used for sharing.

Parameters

exponent_modifier	Unique modifier (guardian id) for exponent [0, Q]
polynomial	Election polynomial
coordinate	The computed coordinate

Returns

0 on success, -1 on failure

elgamal_combine_public_keys()

```
int elgamal_combine_public_keys (
            ElectionKeyPair * guardian,
            ElectionKeyPair * pubkey_map,
            size_t count,
            sp_int * key )
```

Combines multiple ElGamal public keys into a single aggregate key.

This function combines the public keys of a guardian and a set of other guardians into a single aggregate public key. The aggregate key is computed by multiplying all the individual public keys together modulo a large prime.

Parameters

guardian	A pointer to the `ElectionKeyPair` struct of the primary guardian. Its public key will be included in the combination.
pubkey_map	An array of `ElectionKeyPair` structs representing the other guardians whose public keys will be combined.
count	The number of elements in the `pubkey_map` array.
key	A pointer to an `sp_int` where the resulting combined public key will be stored. The caller must allocate memory for this `sp_int` before calling the function.

Returns

0 on success.

-1 on failure (e.g., if any of the multiplication operations fail).

Note

The function assumes that all public keys are valid ElGamal public keys with respect to the same large prime p.

generate_polynomial()

```
int generate_polynomial (
            ElectionPolynomial * polynomial )
```

Generates a polynomial for sharing election keys using Shamir's Secret Sharing.

This function generates a polynomial of a specified degree, where each coefficient is a secret value. The polynomial is used to share an election key among multiple guardians using Shamir's Secret Sharing scheme. Each guardian receives a share of the secret key, which is an evaluation of the polynomial at a specific point. Any subset of guardians above a threshold can reconstruct the original secret key.

For each coefficient of the polynomial, the function generates a random value, computes a commitment to that value, and creates a Schnorr proof of knowledge for the coefficient.

Parameters

polynomial	A pointer to the `ElectionPolynomial` struct where the generated polynomial will be stored. The caller must allocate memory for the `ElectionPolynomial` struct and set the `num_coefficients` field before calling this function. This function will allocate memory for the `coefficients` array within the `ElectionPolynomial` struct, as well as for the `value`, `commitment`, and `proof` fields within each `PolynomialCoefficient` struct.

Returns

0 on success.

-1 on failure (e.g., memory allocation error).

Note

The function allocates memory for the fields within the `polynomial` structure. It is the caller's responsibility to free this memory when the polynomial is no longer needed to prevent memory leaks. See `ElectionPolynomial` and `PolynomialCoefficient` documentation for details.

hash()

```
int hash (
            sp_int * a,
            sp_int * b,
            sp_int * result )
```

Given two mp_ints, calculate their cryptographic hash using SHA256.

Each value is converted to a hexadecimal string and hashed with a delimiter in between (e.g. H(|a|b|)).

Parameters

a	First element
b	Second element
result	The result of the hash

Returns

0 on success, -1 on failure

hash_keys()

```
int hash_keys (
            ElectionKeyPair * guardian,
            ElectionKeyPair * pubkey_map,
            size_t count,
            sp_int * commitment )
```

Computes a SHA256 hash of the commitments from multiple guardians.

This function calculates a SHA256 hash of the commitments made by a primary guardian and a set of other guardians. The hash serves as a binding value that commits all guardians to their respective commitments.

Parameters

guardian	A pointer to the `ElectionKeyPair` struct of the primary guardian. The commitments from its polynomial will be included in the hash.
pubkey_map	An array of `ElectionKeyPair` structs representing the other guardians whose commitments will be included in the hash.
count	The number of elements in the `pubkey_map` array.
commitment	A pointer to an `sp_int` where the resulting hash value will be stored. The caller must allocate memory for this `sp_int` before calling the function.

Returns

0 on success.

-1 on failure (e.g., if SHA256 initialization or update fails, or memory allocation error).

Note

The function concatenates the commitments of all guardians, separated by delimiters, before hashing the result.

hashed_elgamal_decrypt()

```
int hashed_elgamal_decrypt (
            HashedElGamalCiphertext * encrypted_message,
            sp_int * secret_key,
            sp_int * encryption_seed,
            sp_int * message )
```

Decrypt an ElGamal ciphertext using a known ElGamal secret key

Parameters

encrypted_message	struct containing pad, data, and mac
secret_key	The corresponding ElGamal secret key.
encryption_seed	Encryption seed (Q) for election.
message	Decrypted plaintext message.

Returns

0 on success, -1 on failure

hashed_elgamal_encrypt()

```
int hashed_elgamal_encrypt (
            sp_int * message,
            sp_int * nonce,
            sp_int * public_key,
            sp_int * encryption_seed,
            HashedElGamalCiphertext * encrypted_message )
```

Encrypts a message using Hashed ElGamal encryption.

This function encrypts a given message using the Hashed ElGamal encryption scheme. It generates a ciphertext consisting of a pad, data, and MAC (Message Authentication Code).

Parameters

message	The message (sp_int) to be encrypted.
nonce	A random nonce (sp_int) used for encryption.
public_key	The recipient's public key (sp_int).
encryption_seed	An encryption seed (sp_int) used in the key derivation function.
encrypted_message	A pointer to the HashedElGamalCiphertext structure where the resulting ciphertext will be stored.

Returns

0 on success, 1 on memory allocation failure.

make_chaum_pedersen()

```
int make_chaum_pedersen (
            sp_int * alpha,
            sp_int * beta,
            sp_int * secret,
            sp_int * m,
            sp_int * seed,
            sp_int * hash_header,
            ChaumPedersenProof * proof )
```

Generates a Chaum-Pedersen proof to demonstrate that a ciphertext corresponds to a specific plaintext.

This function generates a Chaum-Pedersen proof, which is a zero-knowledge proof that demonstrates that a given ElGamal ciphertext (alpha, beta) encrypts a specific plaintext value (m). The proof shows that the prover knows the secret (nonce) used to create the ciphertext, without revealing the secret itself.

Parameters

alpha	A pointer to the ElGamal ciphertext's alpha component ((g^r \mod p)), which is a large integer.

Parameters

beta	A pointer to the ElGamal ciphertext's beta component ((m \cdot y^r \mod p)), which is a large integer.
secret	A pointer to the nonce ((r)) used to encrypt the message, which is a large integer.
m	A pointer to the plaintext message ((m)), which is a large integer.
seed	A pointer to a seed value used to generate random values within the proof.
hash_header	A pointer to a header value used when generating the challenge, typically the election extended base hash.
proof	A pointer to the `ChaumPedersenProof` struct where the generated proof will be stored. The caller is responsible for allocating memory for the `ChaumPedersenProof` struct. The function will allocate memory for the `pad`, `data`, `challenge`, and `response` fields within the `ChaumPedersenProof` struct.

Returns

0 on success.

-1 on failure (e.g., memory allocation error).

Note

The function allocates memory for the fields within the `proof` structure. It is the caller's responsibility to free this memory when the proof is no longer needed to prevent memory leaks. See `ChaumPedersenProof` documentation for details.

print_byte_array()

```
void print_byte_array (
            const byte * array,
            int size )
```

Prints a byte array to the ESP_LOGI.

This function converts a byte array to a hexadecimal string and prints it using ESP_LOGI.

Parameters

array	The byte array to print.
size	The size of the byte array.

print_sp_int()

```
void print_sp_int (
            sp_int * num )
```

Prints an sp_int to the ESP_LOGI.

This function converts an sp_int to a hexadecimal string and prints it using ESP_LOGI.

Parameters

num	The sp_int to print.

rand_q()

```
int rand_q (
            sp_int * result )
```

Generate a random number below constant Prime Q (small prime)

Parameters

result	The random number

Returns

0 on success, -1 on failure

verify_polynomial_coordinate()

```
int verify_polynomial_coordinate (
            uint8_t * exponent_modifier,
            ElectionPolynomial * polynomial,
            sp_int * coordinate )
```

Verifies that a coordinate lies on the election polynomial.

Given a guardian's identifier, a polynomial, and a coordinate, this function verifies that the coordinate corresponds to a point on the polynomial. This is used to validate partial key backups.

Parameters

exponent_modifier	The unique identifier of the guardian (usually sequence order).
polynomial	The election polynomial to verify against.
coordinate	The coordinate to verify.

Returns

1 if the coordinate is on the polynomial, 0 otherwise. Returns -1 on failure.

3.18 crypto_utils.h

Go to the documentation of this file.

```
#ifndef CRYPTO_UTILS_H
#define CRYPTO_UTILS_H

#include <wolfssl/wolfcrypt/port/Espressif/esp32-crypt.h>
#include <wolfssl/wolfcrypt/wolfmath.h>
```

```
#include <wolfssl/ssl.h>
#include "esp_log.h"
#include "esp_random.h"
#include "constants.h"



#define BLOCK_SIZE 32
#define MODIFIED_CHUNK_SIZE (BLOCK_SIZE + 8)

typedef struct {
    sp_int* pubkey;
    sp_int* commitment;
    sp_int* challenge;
    sp_int* response;
} SchnorrProof;

typedef struct {
    sp_int* value;
    sp_int* commitment;
    SchnorrProof proof;
} Coefficient;

typedef struct {
    int num_coefficients;
    Coefficient* coefficients;
} ElectionPolynomial;

// contains also private key. Be careful when sending!
typedef struct {
    uint8_t guardian_id[6];
    sp_int* public_key;
    sp_int* private_key;
    ElectionPolynomial polynomial;
} ElectionKeyPair;

 typedef struct {
    sp_int* pad;
    sp_int* data;
    sp_int* mac;
} HashedElGamalCiphertext;

 typedef struct {
    uint8_t sender[6];
    uint8_t receiver[6];
    HashedElGamalCiphertext encrypted_coordinate;
 } ElectionPartialKeyPairBackup;


 typedef struct {
    uint8_t sender[6];
    uint8_t receiver[6];
    uint8_t verifier[6];
    bool verified;
 } ElectionPartialKeyVerification;

 typedef struct {
    sp_int* joint_key;
    sp_int* commitment_hash;
 } ElectionJointKey;

typedef struct {
    char* object_id;
    sp_int* ciphertext_pad;
    sp_int* ciphertext_data;
} CiphertextTallySelection;

typedef struct {
    char* object_id;
    int sequence_order;
    sp_int* description_hash;
    int num_selections;
    CiphertextTallySelection* selections;
} CiphertextTallyContest;

typedef struct{
    char* object_id;
    sp_int* base_hash;
    int num_contest;
    CiphertextTallyContest* contests;
} CiphertextTally;

typedef struct {
    sp_int* pad;
    sp_int* data;
    sp_int* challenge;
    sp_int* response;
} ChaumPedersenProof;

typedef struct {
    char* object_id;
    uint8_t guardian_id[6];
    sp_int* decryption;
    ChaumPedersenProof proof;
} CiphertextDecryptionSelection;

typedef struct{
    char* object_id;
    uint8_t guardian_id[6];
    sp_int *public_key;
    sp_int* description_hash;
    int num_selections;
    CiphertextDecryptionSelection* selections;
} CiphertextDecryptionContest;

typedef struct {
    char* object_id;
    uint8_t guardian_id[6];
    sp_int* public_key;
    int num_contest;
    CiphertextDecryptionContest* contests;
} DecryptionShare;

int elgamal_combine_public_keys(ElectionKeyPair *guardian, ElectionKeyPair *pubkey_map, size_t count,
    sp_int *key);
int compute_decryption_share_for_contest(ElectionKeyPair *guardian, CiphertextTallyContest *contest,
    sp_int* base_hash , CiphertextDecryptionContest *dec_contest);

int compute_polynomial_coordinate(uint8_t* exponent_modifier, ElectionPolynomial* polynomial, sp_int
    *coordinate);
int verify_polynomial_coordinate(uint8_t* exponent_modifier, ElectionPolynomial* polynomial, sp_int
    *coordinate);

int hashed_elgamal_encrypt(sp_int *message, sp_int *nonce, sp_int *public_key, sp_int
    *encryption_seed, HashedElGamalCiphertext *encrypted_message);
int hashed_elgamal_decrypt(HashedElGamalCiphertext *encrypted_message, sp_int *secret_key, sp_int
    *encryption_seed, sp_int *message);

int generate_polynomial(ElectionPolynomial *polynomial);

int hash(sp_int *a, sp_int *b, sp_int *result);
int hash_keys(ElectionKeyPair *guardian, ElectionKeyPair *pubkey_map, size_t count, sp_int
    *commitment);
int rand_q(sp_int *result);
int make_chaum_pedersen(sp_int* alpha, sp_int* beta, sp_int* secret, sp_int* m, sp_int* seed, sp_int*
    hash_header, ChaumPedersenProof* proof);

void print_sp_int(sp_int *num);
void print_byte_array(const byte *array, int size);

#endif // CRYPTO_UTILS_H
```

3.19 D:/Dokuments/IoT-election/guardian/components/model/util/include/utils.h File Reference

```
#include <stdint.h>
#include <wolfssl/wolfcrypt/port/Espressif/esp32-crypt.h>
#include <wolfssl/wolfcrypt/wolfmath.h>
#include <wolfssl/ssl.h>
#include "crypto_utils.h"
```

Functions

- void free_DecryptionShare (DecryptionShare *share)

  Frees the memory allocated for a DecryptionShare struct.
- void free_CiphertextTally (CiphertextTally *tally)

  Frees the memory allocated for a CiphertextTally struct.
- void free_CiphertextTallySelection (CiphertextTallySelection *selection)

Frees the memory allocated for a CiphertextTallySelection struct.

- void free_CiphertextTallyContest (CiphertextTallyContest ∗contest)

  Frees the memory allocated for a CiphertextTallyContest struct.
- void free_ChaumPedersenProof (ChaumPedersenProof ∗proof)

  Frees the memory allocated for a ChaumPedersenProof struct.
- void free_CiphertextDecryptionSelection (CiphertextDecryptionSelection ∗selection)

  Frees the memory allocated for a CiphertextDecryptionSelection struct.
- void free_CiphertextDecryptionContest (CiphertextDecryptionContest ∗contest)

  Frees the memory allocated for a CiphertextDecryptionContest struct.
- void free_ElectionPartialKeyPairBackup (ElectionPartialKeyPairBackup ∗backup)

  Frees the memory allocated for an ElectionPartialKeyPairBackup struct.
- void free_ElectionKeyPair (ElectionKeyPair ∗key_pair)

  Frees the memory allocated for an ElectionKeyPair struct.
- void free_ElectionPolynomial (ElectionPolynomial ∗polynomial)

  Frees the memory allocated for an ElectionPolynomial struct.
- void free_Coefficient (Coefficient ∗coefficient)

  Frees the memory allocated for a Coefficient struct.
- void free_SchnorrProof (SchnorrProof ∗proof)

  Frees the memory allocated for a SchnorrProof struct.

3.19.1 Function Documentation

free_ChaumPedersenProof()

```
void free_ChaumPedersenProof (
            ChaumPedersenProof * proof )
```

Frees the memory allocated for a ChaumPedersenProof struct.

This function releases the memory associated with the pad, data, challenge, and response members of the ChaumPedersenProof struct. It also sets these pointers to NULL to prevent double freeing.

Parameters

proof	A pointer to the ChaumPedersenProof struct to free. If proof is NULL, the function returns immediately.

free_CiphertextDecryptionContest()

```
void free_CiphertextDecryptionContest (
            CiphertextDecryptionContest * contest )
```

Frees the memory allocated for a CiphertextDecryptionContest struct.

This function releases the memory associated with the object_id, description_hash, and selections members of the CiphertextDecryptionContest struct. It iterates through the selections array and calls free_CiphertextDecryption↩ Selection for each element before freeing the array itself. It also sets pointers to NULL to prevent double freeing.

Parameters

contest	A pointer to the CiphertextDecryptionContest struct to free. If contest is NULL, the function returns immediately.

free_CiphertextDecryptionSelection()

```
void free_CiphertextDecryptionSelection (
            CiphertextDecryptionSelection * selection )
```

Frees the memory allocated for a CiphertextDecryptionSelection struct.

This function releases the memory associated with the object_id and decryption members of the CiphertextDecryptionSelection struct. It also calls free_ChaumPedersenProof to free the proof member. It sets pointers to NULL to prevent double freeing.

Parameters

selection	A pointer to the CiphertextDecryptionSelection struct to free. If selection is NULL, the function returns immediately.

free_CiphertextTally()

```
void free_CiphertextTally (
            CiphertextTally * tally )
```

Frees the memory allocated for a CiphertextTally struct.

This function releases the memory associated with the object_id, base_hash, and contests members of the CiphertextTally struct. It iterates through the contests array and calls free_CiphertextTallyContest for each element before freeing the array itself. It also sets pointers to NULL to prevent double freeing.

Parameters

tally	A pointer to the CiphertextTally struct to free. If tally is NULL, the function returns immediately.

free_CiphertextTallyContest()

```
void free_CiphertextTallyContest (
            CiphertextTallyContest * contest )
```

Frees the memory allocated for a CiphertextTallyContest struct.

This function releases the memory associated with the object_id, description_hash, and selections members of the CiphertextTallyContest struct. It iterates through the selections array and calls free_CiphertextTallySelection for each element before freeing the array itself. It also sets pointers to NULL to prevent double freeing.

Parameters

contest	A pointer to the CiphertextTallyContest struct to free. If contest is NULL, the function returns immediately.

free_CiphertextTallySelection()

```
void free_CiphertextTallySelection (
            CiphertextTallySelection * selection )
```

Frees the memory allocated for a CiphertextTallySelection struct.

This function releases the memory associated with the object_id, ciphertext_pad, and ciphertext_data members of the CiphertextTallySelection struct. It also sets these pointers to NULL to prevent double freeing.

Parameters

selection	A pointer to the CiphertextTallySelection object to free. If selection is NULL, the function returns immediately.

free_Coefficient()

```
void free_Coefficient (
            Coefficient * coefficient )
```

Frees the memory allocated for a Coefficient struct.

This function releases the memory associated with the commitment and value members of the Coefficient struct. It also calls free_SchnorrProof to free the proof member. It sets pointers to NULL to prevent double freeing.

Parameters

coefficient	A pointer to the Coefficient struct to free. If coefficient is NULL, the function returns immediately.

free_DecryptionShare()

```
void free_DecryptionShare (
            DecryptionShare * share )
```

Frees the memory allocated for a DecryptionShare struct.

This function releases the memory associated with the object_id, public_key, and contests members of the DecryptionShare struct. It iterates through the contests array and calls free_CiphertextDecryptionContest for each element before freeing the array itself. It also sets pointers to NULL to prevent double freeing.

Parameters

share	A pointer to the DecryptionShare struct to free. If share is NULL, the function returns immediately.

free_ElectionKeyPair()

```
void free_ElectionKeyPair (
            ElectionKeyPair * key_pair )
```

Frees the memory allocated for an ElectionKeyPair struct.

This function releases the memory associated with the public_key and private_key members of the ElectionKeyPair struct. It also calls free_ElectionPolynomial to free the polynomial member. It sets pointers to NULL to prevent double freeing. The private key is zeroed out before freeing.

Parameters

key_pair	A pointer to the ElectionKeyPair struct to free. If key_pair is NULL, the function returns immediately.

free_ElectionPartialKeyPairBackup()

```
void free_ElectionPartialKeyPairBackup (
            ElectionPartialKeyPairBackup * backup )
```

Frees the memory allocated for an ElectionPartialKeyPairBackup struct.

This function releases the memory associated with the encrypted_coordinate members of the ElectionPartialKeyPairBackup struct. It also sets pointers to NULL to prevent double freeing.

Parameters

backup	A pointer to the ElectionPartialKeyPairBackup struct to free. If backup is NULL, the function returns immediately.

free_ElectionPolynomial()

```
void free_ElectionPolynomial (
            ElectionPolynomial * polynomial )
```

Frees the memory allocated for an ElectionPolynomial struct.

This function releases the memory associated with the coefficients member of the ElectionPolynomial struct. It iterates through the coefficients array and calls free_Coefficient for each element before freeing the array itself. It also sets the pointer to NULL to prevent double freeing.

Parameters

polynomial	A pointer to the ElectionPolynomial struct to free. If polynomial is NULL, the function returns immediately.

free_SchnorrProof()

```
void free_SchnorrProof (
SchnorrProof * proof )
```

Frees the memory allocated for a SchnorrProof struct.

This function releases the memory associated with the pubkey, commitment, challenge, and response members of the SchnorrProof struct. It also sets pointers to NULL to prevent double freeing.

Parameters

proof	A pointer to the SchnorrProof struct to free. If proof is NULL, the function returns immediately.

3.20 utils.h

Go to the documentation of this file.

```
#ifndef UTILS_H
#define UTILS_H

#include <stdint.h>
#include <wolfssl/wolfcrypt/port/Espressif/esp32-crypt.h>
#include <wolfssl/wolfcrypt/wolfmath.h>
#include <wolfssl/ssl.h>
#include "crypto_utils.h"

void free_DecryptionShare(DecryptionShare* share);
void free_CiphertextTally(CiphertextTally* tally);
void free_CiphertextTallySelection(CiphertextTallySelection* selection);
void free_CiphertextTallyContest(CiphertextTallyContest* contest);
void free_ChaumPedersenProof(ChaumPedersenProof* proof);
void free_CiphertextDecryptionSelection(CiphertextDecryptionSelection* selection);
void free_CiphertextDecryptionContest(CiphertextDecryptionContest* contest);
void free_ElectionPartialKeyPairBackup(ElectionPartialKeyPairBackup* backup);
void free_ElectionKeyPair(ElectionKeyPair *key_pair);
void free_ElectionPolynomial(ElectionPolynomial *polynomial);
void free_Coefficient(Coefficient *coefficient);
void free_SchnorrProof(SchnorrProof *proof);

#endif // UTILS_H
```

3.21 D:/Dokuments/IoT-election/guardian/components/model/util/utils.c File Reference

```
#include "utils.h"
#include <stdlib.h>
```

Functions

- void free_CiphertextTallySelection (CiphertextTallySelection ∗selection)

  Frees the memory allocated for a CiphertextTallySelection struct.
- void free_CiphertextTallyContest (CiphertextTallyContest ∗contest)

  Frees the memory allocated for a CiphertextTallyContest struct.
- void free_CiphertextTally (CiphertextTally ∗tally)

  Frees the memory allocated for a CiphertextTally struct.
- void free_ChaumPedersenProof (ChaumPedersenProof ∗proof)

  Frees the memory allocated for a ChaumPedersenProof struct.
- void free_CiphertextDecryptionSelection (CiphertextDecryptionSelection ∗selection)

  Frees the memory allocated for a CiphertextDecryptionSelection struct.
- void free_CiphertextDecryptionContest (CiphertextDecryptionContest ∗contest)

  Frees the memory allocated for a CiphertextDecryptionContest struct.

- void free_DecryptionShare (DecryptionShare ∗share)

  Frees the memory allocated for a DecryptionShare struct.
- void free_ElectionPartialKeyPairBackup (ElectionPartialKeyPairBackup ∗backup)

  Frees the memory allocated for an ElectionPartialKeyPairBackup struct.
- void free_ElectionKeyPair (ElectionKeyPair ∗key_pair)

  Frees the memory allocated for an ElectionKeyPair struct.
- void free_ElectionPolynomial (ElectionPolynomial ∗polynomial)

  Frees the memory allocated for an ElectionPolynomial struct.
- void free_Coefficient (Coefficient ∗coefficient)

  Frees the memory allocated for a Coefficient struct.
- void free_SchnorrProof (SchnorrProof ∗proof)

  Frees the memory allocated for a SchnorrProof struct.

3.21.1 Function Documentation

free_ChaumPedersenProof()

```
void free_ChaumPedersenProof (
            ChaumPedersenProof ∗ proof )
```

Frees the memory allocated for a ChaumPedersenProof struct.

This function releases the memory associated with the pad, data, challenge, and response members of the ChaumPedersenProof struct. It also sets these pointers to NULL to prevent double freeing.

Parameters

proof	A pointer to the ChaumPedersenProof struct to free. If proof is NULL, the function returns immediately.

free_CiphertextDecryptionContest()

```
void free_CiphertextDecryptionContest (
            CiphertextDecryptionContest ∗ contest )
```

Frees the memory allocated for a CiphertextDecryptionContest struct.

This function releases the memory associated with the object_id, description_hash, and selections members of the CiphertextDecryptionContest struct. It iterates through the selections array and calls free_CiphertextDecryption↵ Selection for each element before freeing the array itself. It also sets pointers to NULL to prevent double freeing.

Parameters

contest	A pointer to the CiphertextDecryptionContest struct to free. If contest is NULL, the function returns immediately.

free_CiphertextDecryptionSelection()

```
void free_CiphertextDecryptionSelection (
            CiphertextDecryptionSelection * selection )
```

Frees the memory allocated for a CiphertextDecryptionSelection struct.

This function releases the memory associated with the object_id and decryption members of the CiphertextDecryptionSelection struct. It also calls free_ChaumPedersenProof to free the proof member. It sets pointers to NULL to prevent double freeing.

Parameters

selection	A pointer to the CiphertextDecryptionSelection struct to free. If selection is NULL, the function returns immediately.

free_CiphertextTally()

```
void free_CiphertextTally (
            CiphertextTally * tally )
```

Frees the memory allocated for a CiphertextTally struct.

This function releases the memory associated with the object_id, base_hash, and contests members of the CiphertextTally struct. It iterates through the contests array and calls free_CiphertextTallyContest for each element before freeing the array itself. It also sets pointers to NULL to prevent double freeing.

Parameters

tally	A pointer to the CiphertextTally struct to free. If tally is NULL, the function returns immediately.

free_CiphertextTallyContest()

```
void free_CiphertextTallyContest (
            CiphertextTallyContest * contest )
```

Frees the memory allocated for a CiphertextTallyContest struct.

This function releases the memory associated with the object_id, description_hash, and selections members of the CiphertextTallyContest struct. It iterates through the selections array and calls free_CiphertextTallySelection for each element before freeing the array itself. It also sets pointers to NULL to prevent double freeing.

Parameters

contest	A pointer to the CiphertextTallyContest struct to free. If contest is NULL, the function returns immediately.

free_CiphertextTallySelection()

```
void free_CiphertextTallySelection (
            CiphertextTallySelection * selection )
```

Frees the memory allocated for a CiphertextTallySelection struct.

This function releases the memory associated with the object_id, ciphertext_pad, and ciphertext_data members of the CiphertextTallySelection struct. It also sets these pointers to NULL to prevent double freeing.

Parameters

selection	A pointer to the CiphertextTallySelection object to free. If selection is NULL, the function returns immediately.

free_Coefficient()

```
void free_Coefficient (
            Coefficient * coefficient )
```

Frees the memory allocated for a Coefficient struct.

This function releases the memory associated with the commitment and value members of the Coefficient struct. It also calls free_SchnorrProof to free the proof member. It sets pointers to NULL to prevent double freeing.

Parameters

coefficient	A pointer to the Coefficient struct to free. If coefficient is NULL, the function returns immediately.

free_DecryptionShare()

```
void free_DecryptionShare (
            DecryptionShare * share )
```

Frees the memory allocated for a DecryptionShare struct.

This function releases the memory associated with the object_id, public_key, and contests members of the DecryptionShare struct. It iterates through the contests array and calls free_CiphertextDecryptionContest for each element before freeing the array itself. It also sets pointers to NULL to prevent double freeing.

Parameters

share	A pointer to the DecryptionShare struct to free. If share is NULL, the function returns immediately.

free_ElectionKeyPair()

```
void free_ElectionKeyPair (
            ElectionKeyPair * key_pair )
```

Frees the memory allocated for an ElectionKeyPair struct.

This function releases the memory associated with the public_key and private_key members of the ElectionKeyPair struct. It also calls free_ElectionPolynomial to free the polynomial member. It sets pointers to NULL to prevent double freeing. The private key is zeroed out before freeing.

Parameters

key_pair	A pointer to the ElectionKeyPair struct to free. If key_pair is NULL, the function returns immediately.

free_ElectionPartialKeyPairBackup()

```
void free_ElectionPartialKeyPairBackup (
            ElectionPartialKeyPairBackup * backup )
```

Frees the memory allocated for an ElectionPartialKeyPairBackup struct.

This function releases the memory associated with the encrypted_coordinate members of the ElectionPartialKeyPairBackup struct. It also sets pointers to NULL to prevent double freeing.

Parameters

backup	A pointer to the ElectionPartialKeyPairBackup struct to free. If backup is NULL, the function returns immediately.

free_ElectionPolynomial()

```
void free_ElectionPolynomial (
            ElectionPolynomial * polynomial )
```

Frees the memory allocated for an ElectionPolynomial struct.

This function releases the memory associated with the coefficients member of the ElectionPolynomial struct. It iterates through the coefficients array and calls free_Coefficient for each element before freeing the array itself. It also sets the pointer to NULL to prevent double freeing.

Parameters

polynomial	A pointer to the ElectionPolynomial struct to free. If polynomial is NULL, the function returns immediately.

free_SchnorrProof()

```
void free_SchnorrProof (
            SchnorrProof * proof )
```

Frees the memory allocated for a SchnorrProof struct.

This function releases the memory associated with the pubkey, commitment, challenge, and response members of the SchnorrProof struct. It also sets pointers to NULL to prevent double freeing.

Parameters

proof	A pointer to the SchnorrProof struct to free. If proof is NULL, the function returns immediately.

3.22 D:/Dokuments/IoT-election/guardian/components/wolfssl/include/user_settings.h File Reference

```
#include "sdkconfig.h"
```

Macros

- #define WOLFSSL_ESPIDF
- #define NO_ESP_SDK_WIFI
- #define NO_SESSION_CACHE
- #define WOLFSSL_SMALL_STACK
- #define DEBUG_WOLFSSL_MALLOC
- #define USE_CERT_BUFFERS_256
- #define USE_CERT_BUFFERS_2048
- #define RSA_LOW_MEM
- #define FP_MAX_BITS 6144
- #define TEST_ESPIDF_ALL_WOLFSSL
- #define WOLFSSL_MD2
- #define HAVE_BLAKE2
- #define HAVE_BLAKE2B
- #define HAVE_BLAKE2S
- #define WC_RC2
- #define WOLFSSL_ALLOW_RC4
- #define HAVE_POLY1305
- #define WOLFSSL_AES_128
- #define WOLFSSL_AES_OFB
- #define WOLFSSL_AES_CFB
- #define WOLFSSL_AES_XTS
- #define WOLFSSL_WOLFSSH
- #define HAVE_AESGCM
- #define WOLFSSL_AES_COUNTER
- #define HAVE_FFDHE
- #define HAVE_FFDHE_2048
- #define WOLFCRYPT_HAVE_SRP
- #define FP_MAX_BITS (8192 $*$ 2)
- #define HAVE_DH
- #define HAVE_DSA
- #define HAVE_HPKE
- #define WOLFSSL_AES_EAX
- #define ECC_SHAMIR
- #define WOLFSSL_AES_SIV
- #define WOLFSSL_CMAC
- #define WOLFSSL_CERT_PIV
- #define SCRYPT_TEST_ALL
- #define HAVE_X963_KDF
- #define NO_SHA
- #define NO_OLD_TLS
- #define NO_DSA
- #define NO_MD4
- #define NO_MD5
- #define NO_PWDBASED
- #define NO_RC4

- #define WOLFSSL_SP_NO_256
- #define NO_CRYPT_BENCHMARK
- #define NO_CRYPT_TEST
- #define NO_PSK
- #define NO_TLS
- #define TFM_TIMING_RESISTANT
- #define BENCH_EMBEDDED
- #define WOLFSSL_TLS13
- #define HAVE_TLS_EXTENSIONS
- #define WC_RSA_PSS
- #define HAVE_HKDF
- #define HAVE_AEAD
- #define HAVE_SUPPORTED_CURVES
- #define WC_KDF_NIST_SP_800_56C
- #define WOLFSSL_BENCHMARK_FIXED_UNITS_KB
- #define NO_FILESYSTEM
- #define NO_OLD_TLS
- #define HAVE_AESGCM
- #define WOLFSSL_SHA512
- #define HAVE_ECC
- #define HAVE_CURVE25519
- #define CURVE25519_SMALL
- #define HAVE_ED25519
- #define OPENSSL_EXTRA
- #define ESP_RSA_TIMEOUT_CNT 0x349F00
- #define HASH_SIZE_LIMIT
- #define USE_FAST_MATH
- #define SP_MATH
- #define WOLFSSL_SP_MATH_ALL
- #define WOLFSSL_SP_SMALL
- #define WOLFSSL_SP_LOW_MEM
- #define WOLFSSL_HAVE_SP_RSA
- #define WOLFSSL_SP_4096
- #define WOLFSSL_SMALL_STACK
- #define HAVE_VERSION_EXTENDED_INFO
- #define HAVE_SESSION_TICKET
- #define WOLFSSL_KEY_GEN
- #define WOLFSSL_CERT_REQ
- #define WOLFSSL_CERT_GEN
- #define WOLFSSL_CERT_EXT
- #define WOLFSSL_SYS_CA_CERTS
- #define WOLFSSL_CERT_TEXT
- #define WOLFSSL_ASN_TEMPLATE
- #define NO_ESP32_CRYPT
- #define NO_WOLFSSL_ESP32_CRYPT_HASH
- #define NO_WOLFSSL_ESP32_CRYPT_AES
- #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
- #define WOLFSSL_ESPIDF_ERROR_PAUSE
- #define NO_HW_MATH_TEST
- #define WOLFSSL_PUBLIC_MP
- #define CTX_CA_CERT ca_cert_der_2048
- #define CTX_CA_CERT_SIZE sizeof_ca_cert_der_2048
- #define CTX_CA_CERT_TYPE WOLFSSL_FILETYPE_ASN1
- #define CTX_SERVER_CERT server_cert_der_2048
- #define CTX_SERVER_CERT_SIZE sizeof_server_cert_der_2048

- #define CTX_SERVER_CERT_TYPE WOLFSSL_FILETYPE_ASN1
- #define CTX_SERVER_KEY server_key_der_2048
- #define CTX_SERVER_KEY_SIZE sizeof_server_key_der_2048
- #define CTX_SERVER_KEY_TYPE WOLFSSL_FILETYPE_ASN1
- #define CTX_CLIENT_CERT client_cert_der_2048
- #define CTX_CLIENT_CERT_SIZE sizeof_client_cert_der_2048
- #define CTX_CLIENT_CERT_TYPE WOLFSSL_FILETYPE_ASN1
- #define CTX_CLIENT_KEY client_key_der_2048
- #define CTX_CLIENT_KEY_SIZE sizeof_client_key_der_2048
- #define CTX_CLIENT_KEY_TYPE WOLFSSL_FILETYPE_ASN1

3.22.1 Macro Definition Documentation

BENCH_EMBEDDED

```
#define BENCH_EMBEDDED
```

CTX_CA_CERT

```
#define CTX_CA_CERT ca_cert_der_2048
```

CTX_CA_CERT_SIZE

```
#define CTX_CA_CERT_SIZE sizeof_ca_cert_der_2048
```

CTX_CA_CERT_TYPE

```
#define CTX_CA_CERT_TYPE WOLFSSL_FILETYPE_ASN1
```

CTX_CLIENT_CERT

```
#define CTX_CLIENT_CERT client_cert_der_2048
```

CTX_CLIENT_CERT_SIZE

```
#define CTX_CLIENT_CERT_SIZE sizeof_client_cert_der_2048
```

CTX_CLIENT_CERT_TYPE

```
#define CTX_CLIENT_CERT_TYPE WOLFSSL_FILETYPE_ASN1
```

CTX_CLIENT_KEY

```
#define CTX_CLIENT_KEY client_key_der_2048
```

CTX_CLIENT_KEY_SIZE

```
#define CTX_CLIENT_KEY_SIZE sizeof_client_key_der_2048
```

CTX_CLIENT_KEY_TYPE

```
#define CTX_CLIENT_KEY_TYPE WOLFSSL_FILETYPE_ASN1
```

CTX_SERVER_CERT

```
#define CTX_SERVER_CERT server_cert_der_2048
```

CTX_SERVER_CERT_SIZE

```
#define CTX_SERVER_CERT_SIZE sizeof_server_cert_der_2048
```

CTX_SERVER_CERT_TYPE

```
#define CTX_SERVER_CERT_TYPE WOLFSSL_FILETYPE_ASN1
```

CTX_SERVER_KEY

```
#define CTX_SERVER_KEY server_key_der_2048
```

CTX_SERVER_KEY_SIZE

```
#define CTX_SERVER_KEY_SIZE sizeof_server_key_der_2048
```

CTX_SERVER_KEY_TYPE

```
#define CTX_SERVER_KEY_TYPE WOLFSSL_FILETYPE_ASN1
```

CURVE25519_SMALL

```
#define CURVE25519_SMALL
```

DEBUG_WOLFSSL_MALLOC

```
#define DEBUG_WOLFSSL_MALLOC
```

ECC_SHAMIR

```
#define ECC_SHAMIR
```

ESP_RSA_TIMEOUT_CNT

```
#define ESP_RSA_TIMEOUT_CNT 0x349F00
```

FP_MAX_BITS [1/2]

```
#define FP_MAX_BITS 6144
```

FP_MAX_BITS [2/2]

```
#define FP_MAX_BITS (8192 * 2)
```

HASH_SIZE_LIMIT

```
#define HASH_SIZE_LIMIT
```

HAVE_AEAD

```
#define HAVE_AEAD
```

HAVE_AESGCM [1/2]

```
#define HAVE_AESGCM
```

HAVE_AESGCM [2/2]

```
#define HAVE_AESGCM
```

HAVE_BLAKE2

```
#define HAVE_BLAKE2
```

HAVE_BLAKE2B

```
#define HAVE_BLAKE2B
```

HAVE_BLAKE2S

```
#define HAVE_BLAKE2S
```

HAVE_CURVE25519

```
#define HAVE_CURVE25519
```

HAVE_DH

```
#define HAVE_DH
```

HAVE_DSA

```
#define HAVE_DSA
```

HAVE_ECC

```
#define HAVE_ECC
```

HAVE_ED25519

```
#define HAVE_ED25519
```

HAVE_FFDHE

```
#define HAVE_FFDHE
```

HAVE_FFDHE_2048

```
#define HAVE_FFDHE_2048
```

HAVE_HKDF

```
#define HAVE_HKDF
```

HAVE_HPKE

```
#define HAVE_HPKE
```

HAVE_POLY1305

```
#define HAVE_POLY1305
```

HAVE_SESSION_TICKET

```
#define HAVE_SESSION_TICKET
```

HAVE_SUPPORTED_CURVES

```
#define HAVE_SUPPORTED_CURVES
```

HAVE_TLS_EXTENSIONS

```
#define HAVE_TLS_EXTENSIONS
```

HAVE_VERSION_EXTENDED_INFO

```
#define HAVE_VERSION_EXTENDED_INFO
```

HAVE_X963_KDF

```
#define HAVE_X963_KDF
```

NO_CRYPT_BENCHMARK

```
#define NO_CRYPT_BENCHMARK
```

NO_CRYPT_TEST

```
#define NO_CRYPT_TEST
```

NO_DSA

```
#define NO_DSA
```

NO_ESP32_CRYPT

```
#define NO_ESP32_CRYPT
```

NO_ESP_SDK_WIFI

```
#define NO_ESP_SDK_WIFI
```

NO_FILESYSTEM

```
#define NO_FILESYSTEM
```

NO_HW_MATH_TEST

```
#define NO_HW_MATH_TEST
```

NO_MD4

```
#define NO_MD4
```

NO_MD5

```
#define NO_MD5
```

NO_OLD_TLS [1/2]

```
#define NO_OLD_TLS
```

NO_OLD_TLS [2/2]

```
#define NO_OLD_TLS
```

NO_PSK

```
#define NO_PSK
```

NO_PWDBASED

```
#define NO_PWDBASED
```

NO_RC4

```
#define NO_RC4
```

NO_SESSION_CACHE

```
#define NO_SESSION_CACHE
```

NO_SHA

```
#define NO_SHA
```

NO_TLS

```
#define NO_TLS
```

NO_WOLFSSL_ESP32_CRYPT_AES

```
#define NO_WOLFSSL_ESP32_CRYPT_AES
```

NO_WOLFSSL_ESP32_CRYPT_HASH

```
#define NO_WOLFSSL_ESP32_CRYPT_HASH
```

NO_WOLFSSL_ESP32_CRYPT_RSA_PRI

```
#define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
```

OPENSSL_EXTRA

```
#define OPENSSL_EXTRA
```

RSA_LOW_MEM

```
#define RSA_LOW_MEM
```

SCRYPT_TEST_ALL

```
#define SCRYPT_TEST_ALL
```

SP_MATH

```
#define SP_MATH
```

TEST_ESPIDF_ALL_WOLFSSL

```
#define TEST_ESPIDF_ALL_WOLFSSL
```

TFM_TIMING_RESISTANT

```
#define TFM_TIMING_RESISTANT
```

USE_CERT_BUFFERS_2048

```
#define USE_CERT_BUFFERS_2048
```

USE_CERT_BUFFERS_256

```
#define USE_CERT_BUFFERS_256
```

USE_FAST_MATH

```
#define USE_FAST_MATH
```

WC_KDF_NIST_SP_800_56C

```
#define WC_KDF_NIST_SP_800_56C
```

WC_RC2

```
#define WC_RC2
```

WC_RSA_PSS

```
#define WC_RSA_PSS
```

WOLFCRYPT_HAVE_SRP

```
#define WOLFCRYPT_HAVE_SRP
```

WOLFSSL_AES_128

```
#define WOLFSSL_AES_128
```

WOLFSSL_AES_CFB

```
#define WOLFSSL_AES_CFB
```

WOLFSSL_AES_COUNTER

```
#define WOLFSSL_AES_COUNTER
```

WOLFSSL_AES_EAX

```
#define WOLFSSL_AES_EAX
```

WOLFSSL_AES_OFB

```
#define WOLFSSL_AES_OFB
```

WOLFSSL_AES_SIV

```
#define WOLFSSL_AES_SIV
```

WOLFSSL_AES_XTS

```
#define WOLFSSL_AES_XTS
```

WOLFSSL_ALLOW_RC4

```
#define WOLFSSL_ALLOW_RC4
```

WOLFSSL_ASN_TEMPLATE

```
#define WOLFSSL_ASN_TEMPLATE
```

WOLFSSL_BENCHMARK_FIXED_UNITS_KB

```
#define WOLFSSL_BENCHMARK_FIXED_UNITS_KB
```

WOLFSSL_CERT_EXT

```
#define WOLFSSL_CERT_EXT
```

WOLFSSL_CERT_GEN

```
#define WOLFSSL_CERT_GEN
```

WOLFSSL_CERT_PIV

```
#define WOLFSSL_CERT_PIV
```

WOLFSSL_CERT_REQ

```
#define WOLFSSL_CERT_REQ
```

WOLFSSL_CERT_TEXT

```
#define WOLFSSL_CERT_TEXT
```

WOLFSSL_CMAC

```
#define WOLFSSL_CMAC
```

WOLFSSL_ESPIDF

```
#define WOLFSSL_ESPIDF
```

WOLFSSL_ESPIDF_ERROR_PAUSE

```
#define WOLFSSL_ESPIDF_ERROR_PAUSE
```

WOLFSSL_HAVE_SP_RSA

```
#define WOLFSSL_HAVE_SP_RSA
```

WOLFSSL_KEY_GEN

```
#define WOLFSSL_KEY_GEN
```

WOLFSSL_MD2

```
#define WOLFSSL_MD2
```

WOLFSSL_PUBLIC_MP

```
#define WOLFSSL_PUBLIC_MP
```

WOLFSSL_SHA512

```
#define WOLFSSL_SHA512
```

WOLFSSL_SMALL_STACK [1/2]

```
#define WOLFSSL_SMALL_STACK
```

WOLFSSL_SMALL_STACK [2/2]

```
#define WOLFSSL_SMALL_STACK
```

WOLFSSL_SP_4096

```
#define WOLFSSL_SP_4096
```

WOLFSSL_SP_LOW_MEM

```
#define WOLFSSL_SP_LOW_MEM
```

WOLFSSL_SP_MATH_ALL

```
#define WOLFSSL_SP_MATH_ALL
```

WOLFSSL_SP_NO_256

```
#define WOLFSSL_SP_NO_256
```

WOLFSSL_SP_SMALL

```
#define WOLFSSL_SP_SMALL
```

WOLFSSL_SYS_CA_CERTS

```
#define WOLFSSL_SYS_CA_CERTS
```

WOLFSSL_TLS13

```
#define WOLFSSL_TLS13
```

WOLFSSL_WOLFSSH

```
#define WOLFSSL_WOLFSSH
```

3.23 user_settings.h

Go to the documentation of this file.

```
/* user_settings.h
 *
 * Copyright (C) 2006-2024 wolfSSL Inc.
 *
 * This file is part of wolfSSL.
 *
 * wolfSSL is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * wolfSSL is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1335, USA
 */

/* This user_settings.h is for Espressif ESP-IDF
 *
 * Standardized wolfSSL Espressif ESP32 + ESP8266 user_settings.h V5.7.0-1
 *
 * Do not include any wolfssl headers here
 *
 * When editing this file:
 * ensure wolfssl_test and wolfssl_benchmark settings match.
 */

/* The Espressif project config file. See also sdkconfig.defaults */
#include "sdkconfig.h"

/* The Espressif sdkconfig will have chipset info.
**
** Some possible values:
**
**   CONFIG_IDF_TARGET_ESP32
**   CONFIG_IDF_TARGET_ESP32S2
**   CONFIG_IDF_TARGET_ESP32S3
**   CONFIG_IDF_TARGET_ESP32C3
**   CONFIG_IDF_TARGET_ESP32C6
*/

#undef  WOLFSSL_ESPIDF
#define WOLFSSL_ESPIDF

/* We don't use WiFi, so don't compile in the esp-sdk-lib WiFi helpers: */
#define NO_ESP_SDK_WIFI

/* Experimental Kyber */
#if 0
    /* Kyber typically needs a minimum 10K stack */
    #define WOLFSSL_EXPERIMENTAL_SETTINGS
    #define WOLFSSL_HAVE_KYBER
    #define WOLFSSL_WC_KYBER
    #define WOLFSSL_SHA3
#endif

/*
 * ONE of these Espressif chip families will be detected from sdkconfig:
 *
 * WOLFSSL_ESP32
 * WOLFSSL_ESP8266
 */
#undef WOLFSSL_ESPWROOM32SE
#undef WOLFSSL_ESP8266
#undef WOLFSSL_ESP32
/* See below for chipset detection from sdkconfig.h */

/* when you want to use SINGLE THREAD. Note Default ESP-IDF is FreeRTOS */
/* #define SINGLE_THREADED */

/* SMALL_SESSION_CACHE saves a lot of RAM for ClientCache and SessionCache.
 * Memory requirement is about 5KB, otherwise 20K is needed when not specified.
 * If extra small footprint is needed, try MICRO_SESSION_CACHE (< 1K)
 * When really desperate or no TLS used, try NO_SESSION_CACHE.  */
#define NO_SESSION_CACHE

/* Small Stack uses more heap. */
#define WOLFSSL_SMALL_STACK

/* Full debugging turned off, but show malloc failure detail */
/* #define DEBUG_WOLFSSL */
#define DEBUG_WOLFSSL_MALLOC

/* See test.c that sets cert buffers; we'll set them here: */
#define USE_CERT_BUFFERS_256
#define USE_CERT_BUFFERS_2048

/* RSA_LOW_MEM: Half as much memory but twice as slow. */
#define RSA_LOW_MEM

// * FP_MAX_BITS defaults to 4096 bits. This allows only multiplication of up to 2048x2048 bits.
#define FP_MAX_BITS 6144

/* Uncommon settings for testing only */
#define TEST_ESPIDF_ALL_WOLFSSL
#ifdef  TEST_ESPIDF_ALL_WOLFSSL
    #define WOLFSSL_MD2
    #define HAVE_BLAKE2
    #define HAVE_BLAKE2B
    #define HAVE_BLAKE2S

    #define WC_RC2
    #define WOLFSSL_ALLOW_RC4

    #define HAVE_POLY1305

    #define WOLFSSL_AES_128
    #define WOLFSSL_AES_OFB
    #define WOLFSSL_AES_CFB
    #define WOLFSSL_AES_XTS

    /* #define WC_SRTP_KDF */
    /* TODO Causes failure with Espressif AES HW Enabled */
    /* #define HAVE_AES_ECB */
    /* #define HAVE_AESCCM  */
    /* TODO sanity check when missing HAVE_AES_ECB */
    #define WOLFSSL_WOLFSSH

    #define HAVE_AESGCM
    #define WOLFSSL_AES_COUNTER

    #define HAVE_FFDHE
    #define HAVE_FFDHE_2048
    #if defined(CONFIG_IDF_TARGET_ESP8266)
        /* TODO Full size SRP is disabled on the ESP8266 at this time.
         * Low memory issue? */
        #define WOLFCRYPT_HAVE_SRP
        /* MIN_FFDHE_FP_MAX_BITS = (MIN_FFDHE_BITS * 2); see settings.h */
        #define FP_MAX_BITS MIN_FFDHE_FP_MAX_BITS
    #elif defined(CONFIG_IDF_TARGET_ESP32)   || \
          defined(CONFIG_IDF_TARGET_ESP32S2) || \
          defined(CONFIG_IDF_TARGET_ESP32S3)
        #define WOLFCRYPT_HAVE_SRP
        #define FP_MAX_BITS (8192 * 2)
    #elif defined(CONFIG_IDF_TARGET_ESP32C3) || \
          defined(CONFIG_IDF_TARGET_ESP32H2)
        /* SRP Known to be working on this target::*/
        #define WOLFCRYPT_HAVE_SRP
        #define FP_MAX_BITS (8192 * 2)
    #else
        /* For everything else, give a try and see if SRP working: */
        #define WOLFCRYPT_HAVE_SRP
        #define FP_MAX_BITS (8192 * 2)
    #endif

    #define HAVE_DH

    /* TODO: there may be a problem with HAVE_CAMELLIA with HW AES disabled.
```

```
     * Do not define NO_WOLFSSL_ESP32_CRYPT_AES when enabled: */
    /* #define HAVE_CAMELLIA */

    /* DSA requires old SHA */
    #define HAVE_DSA

    /* Needs SHA512 ? */
    #define HAVE_HPKE

    /* Not for Espressif? */
    #if defined(CONFIG_IDF_TARGET_ESP32C2) || \
        defined(CONFIG_IDF_TARGET_ESP8684) || \
        defined(CONFIG_IDF_TARGET_ESP32H2) || \
        defined(CONFIG_IDF_TARGET_ESP8266)

        #if defined(CONFIG_IDF_TARGET_ESP8266)
            #undef HAVE_ECC
            #undef HAVE_ECC_CDH
            #undef HAVE_CURVE25519

            /* TODO does CHACHA also need alignment? Failing on ESP8266
             * See SHA256 __attribute__((aligned(4))); and WC_SHA256_ALIGN */
            #ifdef HAVE_CHACHA
                #error "HAVE_CHACHA not supported on ESP8266"
            #endif
            #ifdef HAVE_XCHACHA
                #error "HAVE_XCHACHA not supported on ESP8266"
            #endif
        #else
            #define HAVE_XCHACHA
            #define HAVE_CHACHA
            /* TODO Not enabled at this time, needs further testing:
             *   #define WC_SRTP_KDF
             *   #define HAVE_COMP_KEY
             *   #define WOLFSSL_HAVE_XMSS
             */
        #endif
        /* TODO AES-EAX not working on this platform */

        /* Optionally disable DH
         *   #undef HAVE_DH
         *   #undef HAVE_FFDHE
         */

        /* ECC_SHAMIR out of memory on ESP32-C2 during ECC  */
        #ifndef HAVE_ECC
            #define ECC_SHAMIR
        #endif
    #else
        #define WOLFSSL_AES_EAX

        #define ECC_SHAMIR
    #endif

    /* Only for WOLFSSL_IMX6_CAAM / WOLFSSL_QNX_CAAM ? */
    /* #define WOLFSSL_CAAM      */
    /* #define WOLFSSL_CAAM_BLOB */

    #define WOLFSSL_AES_SIV
    #define WOLFSSL_CMAC

    #define WOLFSSL_CERT_PIV

    /* HAVE_SCRYPT may turn on HAVE_PBKDF2 see settings.h */
    /* #define HAVE_SCRYPT */
    #define SCRYPT_TEST_ALL
    #define HAVE_X963_KDF
#endif

/* optionally turn off SHA512/224 SHA512/256 */
/* #define WOLFSSL_NOSHA512_224 */
/* #define WOLFSSL_NOSHA512_256 */
// Wolfssl was deprecated in ESP-TLS.
//FREERTOS_ENABLE_BACKWARD_COMPATIBILITY

/* when you want to use SINGLE THREAD. Note Default ESP-IDF is FreeRTOS */
/* #define SINGLE_THREADED */

/* Turning off features reducing library size */
#define NO_SHA
#define NO_OLD_TLS
#define NO_DSA
#define NO_MD4
#define NO_MD5
#define NO_PWDBASED
#define NO_RC4
#define WOLFSSL_SP_NO_256
```

```
#define NO_CRYPT_BENCHMARK
#define NO_CRYPT_TEST
#define NO_PSK
#define NO_TLS

// Hardening Enable TFM Timing Resistant Code
#define TFM_TIMING_RESISTANT

#define BENCH_EMBEDDED

/* TLS 1.3                                 */
#define WOLFSSL_TLS13
#define HAVE_TLS_EXTENSIONS
#define WC_RSA_PSS
#define HAVE_HKDF
#define HAVE_AEAD
#define HAVE_SUPPORTED_CURVES
#define WC_KDF_NIST_SP_800_56C

#define WOLFSSL_BENCHMARK_FIXED_UNITS_KB

#define NO_FILESYSTEM

#define NO_OLD_TLS

#define HAVE_AESGCM

/* Optional RIPEMD: RACE Integrity Primitives Evaluation Message Digest */
/* #define WOLFSSL_RIPEMD */

/* when you want to use SHA224 */
//#define WOLFSSL_SHA224 ++

/* when you want to use SHA384 */
//#define WOLFSSL_SHA384 ++

/* when you want to use SHA512 */
#define WOLFSSL_SHA512

/* when you want to use SHA3 */
//#define WOLFSSL_SHA3 ++

 /* ED25519 requires SHA512 */
//#define HAVE_ED25519 ++

/* Some features not enabled for ESP8266: */
#if defined(CONFIG_IDF_TARGET_ESP8266) || \
    defined(CONFIG_IDF_TARGET_ESP32C2)
    /* TODO determine low memory configuration for ECC. */
#else
    #define HAVE_ECC
    #define HAVE_CURVE25519
    #define CURVE25519_SMALL
#endif

#define HAVE_ED25519

/* Optional OPENSSL compatibility */
#define OPENSSL_EXTRA

/* #Optional HAVE_PKCS7 */
//#define HAVE_PKCS7 ++

#if defined(HAVE_PKCS7)
    /* HAVE_PKCS7 may enable HAVE_PBKDF2 see settings.h */
    #define NO_PBKDF2

    #define HAVE_AES_KEYWRAP
    #define HAVE_X963_KDF
    #define WOLFSSL_AES_DIRECT
#endif

/* when you want to use AES counter mode */
/* #define WOLFSSL_AES_DIRECT */
/* #define WOLFSSL_AES_COUNTER */

/* esp32-wroom-32se specific definition */
#if defined(WOLFSSL_ESPWROOM32SE)
    #define WOLFSSL_ATECC508A
    #define HAVE_PK_CALLBACKS
    /* when you want to use a custom slot allocation for ATECC608A */
    /* unless your configuration is unusual, you can use default   */
    /* implementation.                                             */
    /* #define CUSTOM_SLOT_ALLOCATION                              */
#endif

/* WC_NO_CACHE_RESISTANT: slower but more secure */
/* #define WC_NO_CACHE_RESISTANT */

/* TFM_TIMING_RESISTANT: slower but more secure */
/* #define TFM_TIMING_RESISTANT */

/* #define WOLFSSL_ATECC508A_DEBUG         */

/* date/time                               */
/* if it cannot adjust time in the device, */
/* enable macro below                      */
/* #define NO_ASN_TIME */
/* #define XTIME time */


/* adjust wait-timeout count if you see timeout in RSA HW acceleration */
#define ESP_RSA_TIMEOUT_CNT    0x349F00

/* hash limit for test.c */
#define HASH_SIZE_LIMIT

/* USE_FAST_MATH is default */
#define USE_FAST_MATH

/*****      Use SP_MATH      *****/
#undef USE_FAST_MATH           //+-
#define SP_MATH                //+-
#define WOLFSSL_SP_MATH_ALL  //+-

#define WOLFSSL_SP_SMALL //++
#define WOLFSSL_SP_LOW_MEM //++
#define WOLFSSL_HAVE_SP_RSA //++
#define WOLFSSL_SP_4096 //++
/* #define WOLFSSL_SP_RISCV32    */

/***** Use Integer Heap Math *****/
/* #undef USE_FAST_MATH          */
/* #define USE_INTEGER_HEAP_MATH */


#define WOLFSSL_SMALL_STACK


#define HAVE_VERSION_EXTENDED_INFO
/* #define HAVE_WC_INTROSPECTION */

#define  HAVE_SESSION_TICKET

/* #define HAVE_HASHDRBG */

#define WOLFSSL_KEY_GEN
#define WOLFSSL_CERT_REQ
#define WOLFSSL_CERT_GEN
#define WOLFSSL_CERT_EXT
#define WOLFSSL_SYS_CA_CERTS


#define WOLFSSL_CERT_TEXT

#define WOLFSSL_ASN_TEMPLATE

/*
#undef  WOLFSSL_KEY_GEN
#undef  WOLFSSL_CERT_REQ
#undef  WOLFSSL_CERT_GEN
#undef  WOLFSSL_CERT_EXT
#undef  WOLFSSL_SYS_CA_CERTS
*/

/* command-line options
--enable-keygen
--enable-certgen
--enable-certreq
--enable-certext
--enable-asn-template
*/

/* Chipset detection from sdkconfig.h
 * Default is HW enabled unless turned off.
 * Uncomment lines to force SW instead of HW acceleration */
#if defined(CONFIG_IDF_TARGET_ESP32)
    #define WOLFSSL_ESP32
    /*  Alternatively, if there's an ECC Secure Element present: */
    /* #define WOLFSSL_ESPWROOM32SE */

    /* wolfSSL HW Acceleration supported on ESP32. Uncomment to disable: */
    /* #define NO_ESP32_CRYPT  */
    /* #define NO_WOLFSSL_ESP32_CRYPT_HASH  */
    /* #define NO_WOLFSSL_ESP32_CRYPT_AES */
    /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI */
    /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL */
    /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD  */
    /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */

    /*  These are defined automatically in esp32-crypt.h, here for clarity:  */
    #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA224 /* no SHA224 HW on ESP32  */

    #undef  ESP_RSA_MULM_BITS
    #define ESP_RSA_MULM_BITS 16 /* TODO add compile-time warning */
    /***** END CONFIG_IDF_TARGET_ESP32 *****/

#elif defined(CONFIG_IDF_TARGET_ESP32S2)
    #define WOLFSSL_ESP32
    /* wolfSSL HW Acceleration supported on ESP32-S2. Uncomment to disable: */
    /*  #define NO_ESP32_CRYPT                 */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_HASH    */
    /* Note: There's no AES192 HW on the ESP32-S2; falls back to SW */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_AES     */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL  */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD  */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */
    /***** END CONFIG_IDF_TARGET_ESP32S2 *****/

#elif defined(CONFIG_IDF_TARGET_ESP32S3)
    #define WOLFSSL_ESP32
    /* wolfSSL HW Acceleration supported on ESP32-S3. Uncomment to disable: */
    /*  #define NO_ESP32_CRYPT                         */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_HASH            */
    /* Note: There's no AES192 HW on the ESP32-S3; falls back to SW */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_AES             */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI         */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL  */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD  */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */
    /***** END CONFIG_IDF_TARGET_ESP32S3 *****/

#elif defined(CONFIG_IDF_TARGET_ESP32C2) || \
      defined(CONFIG_IDF_TARGET_ESP8684)
    #define WOLFSSL_ESP32
    /* ESP8684 is essentially ESP32-C2 chip + flash embedded together in a
     * single QFN 4x4 mm package. Out of released documentation, Technical
     * Reference Manual as well as ESP-IDF Programming Guide is applicable
     * to both ESP32-C2 and ESP8684.
     *
     * See:
https://www.esp32.com/viewtopic.php?f=5&t=27926#:~:text=ESP8684%20is%20essentially%20ESP32%2DC2,both%20ESP32%2DC2%20and
*/

    /* wolfSSL HW Acceleration supported on ESP32-C2. Uncomment to disable: */
    /*  #define NO_ESP32_CRYPT                 */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_HASH    */ /* to disable all SHA HW   */

    /* These are defined automatically in esp32-crypt.h, here for clarity    */
    #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA384    /* no SHA384 HW on C2  */
    #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA512    /* no SHA512 HW on C2  */

    /* There's no AES or RSA/Math accelerator on the ESP32-C2
     * Auto defined with NO_WOLFSSL_ESP32_CRYPT_RSA_PRI, for clarity: */
    #define NO_WOLFSSL_ESP32_CRYPT_AES
    #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
    #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL
    #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD
    #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD
    /***** END CONFIG_IDF_TARGET_ESP32C2 *****/

#elif defined(CONFIG_IDF_TARGET_ESP32C3)
    #define WOLFSSL_ESP32
    /* wolfSSL HW Acceleration supported on ESP32-C3. Uncomment to disable: */

    /*  #define NO_ESP32_CRYPT                 */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_HASH    */ /* to disable all SHA HW   */

    /* These are defined automatically in esp32-crypt.h, here for clarity:  */
    #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA384    /* no SHA384 HW on C6  */
    #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA512    /* no SHA512 HW on C6  */

    /*  #define NO_WOLFSSL_ESP32_CRYPT_AES             */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI         */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL  */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD  */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */
    /***** END CONFIG_IDF_TARGET_ESP32C3 *****/

#elif defined(CONFIG_IDF_TARGET_ESP32C6)
```

```
    #define WOLFSSL_ESP32
    /* wolfSSL HW Acceleration supported on ESP32-C6. Uncomment to disable: */

    /*  #define NO_ESP32_CRYPT                 */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_HASH    */
    /*  These are defined automatically in esp32-crypt.h, here for clarity:  */
    #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA384    /* no SHA384 HW on C6  */
    #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA512    /* no SHA512 HW on C6  */

    /*  #define NO_WOLFSSL_ESP32_CRYPT_AES             */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI         */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL  */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD  */
    /*  #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */
    /***** END CONFIG_IDF_TARGET_ESP32C6 *****/

#elif defined(CONFIG_IDF_TARGET_ESP32H2)
    #define WOLFSSL_ESP32
    /*  wolfSSL Hardware Acceleration not yet implemented */
    #define NO_ESP32_CRYPT
    #define NO_WOLFSSL_ESP32_CRYPT_HASH
    #define NO_WOLFSSL_ESP32_CRYPT_AES
    #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
    /***** END CONFIG_IDF_TARGET_ESP32H2 *****/

#elif defined(CONFIG_IDF_TARGET_ESP8266)
    #define WOLFSSL_ESP8266

    /* There's no hardware encryption on the ESP8266 */
    /* Consider using the ESP32-C2/C3/C6
     * See https://www.espressif.com/en/products/socs/esp32-c2 */
    #define NO_ESP32_CRYPT
    #define NO_WOLFSSL_ESP32_CRYPT_HASH
    #define NO_WOLFSSL_ESP32_CRYPT_AES
    #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
    /***** END CONFIG_IDF_TARGET_ESP266 *****/

#elif defined(CONFIG_IDF_TARGET_ESP8684)
    /*  There's no Hardware Acceleration available on ESP8684 */
    #define NO_ESP32_CRYPT
    #define NO_WOLFSSL_ESP32_CRYPT_HASH
    #define NO_WOLFSSL_ESP32_CRYPT_AES
    #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
    /***** END CONFIG_IDF_TARGET_ESP8684 *****/

#else
    /* Anything else encountered, disable HW accleration */
    #warning "Unexpected CONFIG_IDF_TARGET_NN value"
    #define NO_ESP32_CRYPT
    #define NO_WOLFSSL_ESP32_CRYPT_HASH
    #define NO_WOLFSSL_ESP32_CRYPT_AES
    #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
#endif /* CONFIG_IDF_TARGET Check */

/* RSA primitive specific definition, listed AFTER the Chipset detection */
#if defined(WOLFSSL_ESP32) || defined(WOLFSSL_ESPWROOM32SE)
    /* Consider USE_FAST_MATH and SMALL_STACK                        */

    #ifndef NO_RSA
        #define ESP32_USE_RSA_PRIMITIVE

        #if defined(CONFIG_IDF_TARGET_ESP32)
            #ifdef CONFIG_ESP_MAIN_TASK_STACK_SIZE
                #if CONFIG_ESP_MAIN_TASK_STACK_SIZE < 10500
                    #warning "RSA may be difficult with less than 10KB Stack "/
                #endif
            #endif

            /* NOTE HW unreliable for small values! */
            /* threshold for performance adjustment for HW primitive use   */
            /* X bits of G^X mod P greater than                            */
            #undef  ESP_RSA_EXPT_XBITS
            #define ESP_RSA_EXPT_XBITS 32

            /* X and Y of X * Y mod P greater than                         */
            #undef  ESP_RSA_MULM_BITS
            #define ESP_RSA_MULM_BITS  16
        #endif
    #endif
#endif


/* Pause in a loop rather than exit. */
#define WOLFSSL_ESPIDF_ERROR_PAUSE

#define NO_HW_MATH_TEST

```

```
/* for test.c */
/* #define HASH_SIZE_LIMIT */

/* Optionally turn off HW math checks */
/* #define NO_HW_MATH_TEST */

/* Optionally include alternate HW test library: alt_hw_test.h */
/* When enabling, the ./components/wolfssl/CMakeLists.txt file
 * will need the name of the library in the idf_component_register
 * for the PRIV_REQUIRES list. */
/* #define INCLUDE_ALT_HW_TEST */

/* optionally turn off individual math HW acceleration features */

/* Turn off Large Number ESP32 HW Multiplication:
** [Z = X * Y] in esp_mp_mul()                                  */
/* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL                */

/* Turn off Large Number ESP32 HW Modular Exponentiation:
** [Z = X^Y mod M] in esp_mp_exptmod()                          */
/* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD               */

/* Turn off Large Number ESP32 HW Modular Multiplication
** [Z = X * Y mod M] in esp_mp_mulmod()                         */
/* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD                */


/* used by benchmark: */
#define WOLFSSL_PUBLIC_MP

/* when turning on ECC508 / ECC608 support
#define WOLFSSL_ESPWROOM32SE
#define HAVE_PK_CALLBACKS
#define WOLFSSL_ATECC508A
#define ATCA_WOLFSSL
*/

/***************************** Certificate Macros *****************************
 *
 * The section below defines macros used in typically all of the wolfSSL
 * examples such as the client and server for certs stored in header files.
 *
 * There are various certificate examples in this header file:
 * https://github.com/wolfSSL/wolfssl/blob/master/wolfssl/certs_test.h
 *
 * To use the sets of macros below, define *one* of these:
 *
 *    USE_CERT_BUFFERS_1024  - ECC 1024 bit encoded ASN1
 *    USE_CERT_BUFFERS_2048  - RSA 2048 bit encoded ASN1
 *    WOLFSSL_SM[2,3,4]      - SM Ciphers
 *
 * For example: define USE_CERT_BUFFERS_2048 to use CA Certs used in this
 *  wolfSSL function for the `ca_cert_der_2048` buffer, size and types:
 *
 *     ret = wolfSSL_CTX_load_verify_buffer(ctx,
 *                                          CTX_CA_CERT,
 *                                          CTX_CA_CERT_SIZE,
 *                                          CTX_CA_CERT_TYPE);
 *
 * See
https://www.wolfssl.com/documentation/manuals/wolfssl/group__CertsKeys.html#function-wolfssl_ctx_load_verify_buffer
 *
 * In this case the CTX_CA_CERT will be defined as `ca_cert_der_2048` as
 * defined here: https://github.com/wolfSSL/wolfssl/blob/master/wolfssl/certs_test.h
 *
 * The CTX_CA_CERT_SIZE and CTX_CA_CERT_TYPE are similarly used to reference
 * array size and cert type respectively.
 *
 * Similarly for loading the private client key:
 *
 *  ret = wolfSSL_CTX_use_PrivateKey_buffer(ctx,
 *                                          CTX_CLIENT_KEY,
 *                                          CTX_CLIENT_KEY_SIZE,
 *                                          CTX_CLIENT_KEY_TYPE);
 *
 * see
https://www.wolfssl.com/documentation/manuals/wolfssl/group__CertsKeys.html#function-wolfssl_ctx_use_privatekey_buffer
 *
 * Similarly, the other macros are for server certificates and keys:
 *   `CTX_SERVER_CERT` and `CTX_SERVER_KEY` are available.
 *
 * The certificate and key names are typically `static const unsigned char`
 * arrays. The [NAME]_size are typically `sizeof([array name])`, and the types
 * are the known wolfSSL encoding type integers (e.g. WOLFSSL_FILETYPE_PEM).
 *
 * See `SSL_FILETYPE_[name]` in
 *   https://github.com/wolfSSL/wolfssl/blob/master/wolfssl/ssl.h
 *
 * See Abstract Syntax Notation One (ASN.1) in:
 *   https://github.com/wolfSSL/wolfssl/blob/master/wolfssl/wolfcrypt/asn.h
 *
 * Optional SM4 Ciphers:
 *
 * Although the SM ciphers are shown here, the `certs_test_sm.h` may not yet
 * be available. See:
 *   https://github.com/wolfSSL/wolfssl/pull/6825
 *   https://github.com/wolfSSL/wolfsm
 *
 * Uncomment these 3 macros to enable the SM Ciphers and use the macros below.
 */

/*
#define WOLFSSL_SM2
#define WOLFSSL_SM3
#define WOLFSSL_SM4
*/

/* Conditional macros used in wolfSSL TLS client and server examples */
#if defined(WOLFSSL_SM2) || defined(WOLFSSL_SM3) || defined(WOLFSSL_SM4)
    #include <wolfssl/certs_test_sm.h>
    #define CTX_CA_CERT          root_sm2
    #define CTX_CA_CERT_SIZE     sizeof_root_sm2
    #define CTX_CA_CERT_TYPE     WOLFSSL_FILETYPE_PEM
    #define CTX_SERVER_CERT      server_sm2
    #define CTX_SERVER_CERT_SIZE sizeof_server_sm2
    #define CTX_SERVER_CERT_TYPE WOLFSSL_FILETYPE_PEM
    #define CTX_SERVER_KEY       server_sm2_priv
    #define CTX_SERVER_KEY_SIZE  sizeof_server_sm2_priv
    #define CTX_SERVER_KEY_TYPE  WOLFSSL_FILETYPE_PEM

    #undef  WOLFSSL_BASE16
    #define WOLFSSL_BASE16
#else
    #if defined(USE_CERT_BUFFERS_2048)
        /* Be sure to include in app when using example certs: */
        /* #include <wolfssl/certs_test.h>                      */
        #define CTX_CA_CERT          ca_cert_der_2048
        #define CTX_CA_CERT_SIZE     sizeof_ca_cert_der_2048
        #define CTX_CA_CERT_TYPE     WOLFSSL_FILETYPE_ASN1

        #define CTX_SERVER_CERT      server_cert_der_2048
        #define CTX_SERVER_CERT_SIZE sizeof_server_cert_der_2048
        #define CTX_SERVER_CERT_TYPE WOLFSSL_FILETYPE_ASN1
        #define CTX_SERVER_KEY       server_key_der_2048
        #define CTX_SERVER_KEY_SIZE  sizeof_server_key_der_2048
        #define CTX_SERVER_KEY_TYPE  WOLFSSL_FILETYPE_ASN1

        #define CTX_CLIENT_CERT      client_cert_der_2048
        #define CTX_CLIENT_CERT_SIZE sizeof_client_cert_der_2048
        #define CTX_CLIENT_CERT_TYPE WOLFSSL_FILETYPE_ASN1
        #define CTX_CLIENT_KEY       client_key_der_2048
        #define CTX_CLIENT_KEY_SIZE  sizeof_client_key_der_2048
        #define CTX_CLIENT_KEY_TYPE  WOLFSSL_FILETYPE_ASN1

    #elif defined(USE_CERT_BUFFERS_1024)
        /* Be sure to include in app when using example certs: */
        /* #include <wolfssl/certs_test.h>                      */
        #define CTX_CA_CERT          ca_cert_der_1024
        #define CTX_CA_CERT_SIZE     sizeof_ca_cert_der_1024
        #define CTX_CA_CERT_TYPE     WOLFSSL_FILETYPE_ASN1

        #define CTX_CLIENT_CERT      client_cert_der_1024
        #define CTX_CLIENT_CERT_SIZE sizeof_client_cert_der_1024
        #define CTX_CLIENT_CERT_TYPE WOLFSSL_FILETYPE_ASN1
        #define CTX_CLIENT_KEY       client_key_der_1024
        #define CTX_CLIENT_KEY_SIZE  sizeof_client_key_der_1024
        #define CTX_CLIENT_KEY_TYPE  WOLFSSL_FILETYPE_ASN1

        #define CTX_SERVER_CERT      server_cert_der_1024
        #define CTX_SERVER_CERT_SIZE sizeof_server_cert_der_1024
        #define CTX_SERVER_CERT_TYPE WOLFSSL_FILETYPE_ASN1
        #define CTX_SERVER_KEY       server_key_der_1024
        #define CTX_SERVER_KEY_SIZE  sizeof_server_key_der_1024
        #define CTX_SERVER_KEY_TYPE  WOLFSSL_FILETYPE_ASN1
    #else
        /* Optionally define custom cert arrays, sizes, and types here */
        #error "Must define USE_CERT_BUFFERS_2048 or USE_CERT_BUFFERS_1024"
    #endif
#endif /* Conditional key and cert constant names */

/******************************************************************************
** Sanity Checks
******************************************************************************/
#if defined(CONFIG_ESP_MAIN_TASK_STACK_SIZE)
```

```
    #if defined(WOLFCRYPT_HAVE_SRP)
        #if defined(FP_MAX_BITS)
            #if FP_MAX_BITS <  (8192 * 2)
                #define ESP_SRP_MINIMUM_STACK_8K (24 * 1024)
            #else
                #define ESP_SRP_MINIMUM_STACK_8K (28 * 1024)
            #endif
        #else
            #error "Please define FP_MAX_BITS when using WOLFCRYPT_HAVE_SRP."
        #endif

        #if (CONFIG_ESP_MAIN_TASK_STACK_SIZE < ESP_SRP_MINIMUM_STACK)
            #warning "WOLFCRYPT_HAVE_SRP enabled with small stack size"
        #endif
    #endif
#else
    #warning "CONFIG_ESP_MAIN_TASK_STACK_SIZE not defined!"
#endif
```

3.24 D:/Dokuments/IoT-election/guardian/main/main.c File Reference

```
#include "adapter.h"
#include "nvs_flash.h"
#include "test_performance.h"
#include "esp_task_wdt.h"
```

Functions

- void app_main (void)

3.24.1 Function Documentation

app_main()

```
void app_main (
            void )
```

3.25 D:/Dokuments/IoT-election/guardian/main/test_performance.c File Reference

```
#include "test_performance.h"
```

Macros

- #define MEASUREMENT_RUNS 30

Functions

- void perform_measurements_keygen (int quorum)

  Performs timing measurements for election key pair generation and calculates statistics.
- void perform_measurements_backup ()

  Performs timing measurements for election partial key backup generation and calculates statistics.
- void perform_measurements_verification ()

  Performs timing measurements for election partial key backup verification and calculates statistics.

3.25.1 Macro Definition Documentation

MEASUREMENT_RUNS

```
#define MEASUREMENT_RUNS 30
```

3.25.2 Function Documentation

perform_measurements_backup()

```
void perform_measurements_backup ( )
```

Performs timing measurements for election partial key backup generation and calculates statistics.

perform_measurements_keygen()

```
void perform_measurements_keygen (
            int quorum )
```

Performs timing measurements for election key pair generation and calculates statistics.

Parameters

quorum	The quorum size used for key generation.

perform_measurements_verification()

```
void perform_measurements_verification ( )
```

Performs timing measurements for election partial key backup verification and calculates statistics.

3.26 D:/Dokuments/IoT-election/guardian/main/test_performance.h File Reference

```
#include "test_performance.h"
#include "model.h"
#include "crypto_utils.h"
#include "freertos/task.h"
#include "freertos/FreeRTOS.h"
#include "esp_task_wdt.h"
#include "esp_timer.h"
#include <math.h>
#include "sdkconfig.h"
```

Functions

- void perform_measurements_keygen (int quorum)

  Performs timing measurements for election key pair generation and calculates statistics.
- void perform_measurements_backup ()

  Performs timing measurements for election partial key backup generation and calculates statistics.
- void perform_measurements_verification ()

  Performs timing measurements for election partial key backup verification and calculates statistics.

3.26.1 Function Documentation

perform_measurements_backup()

```
void perform_measurements_backup ( )
```

Performs timing measurements for election partial key backup generation and calculates statistics.

perform_measurements_keygen()

```
void perform_measurements_keygen (
              int quorum )
```

Performs timing measurements for election key pair generation and calculates statistics.

Parameters

quorum	The quorum size used for key generation.

perform_measurements_verification()

```
void perform_measurements_verification ( )
```

Performs timing measurements for election partial key backup verification and calculates statistics.

3.27 test_performance.h

Go to the documentation of this file.

```
#ifndef TEST_PERFORMANCE_H
#define TEST_PERFORMANCE_H
#include "test_performance.h"
#include "model.h"
#include "crypto_utils.h"
#include "freertos/task.h"
#include "freertos/FreeRTOS.h"
#include "esp_task_wdt.h"
#include "esp_timer.h"
#include <math.h>
#include "sdkconfig.h"
#include "esp_task_wdt.h"

void perform_measurements_keygen(int quorum);
void perform_measurements_backup();
void perform_measurements_verification();

#endif // TEST_PERFORMANCE_H
```

3.28 D:/Dokuments/IoT-election/guardian/README.md File Reference

Index