

ESP32 Guardian Component

1

Generated by Doxygen 1.10.0

1 Guardian - ElectionGuard ESP32 Client	1
1.1 Overview	1
1.2 Project Structure	1
1.3 Functionality	2
1.4 Building and Flashing	2
1.5 Testing and Performance Evaluation	2
2 File Index	3
2.1 File List	3
3 File Documentation	3
3.1 D:/Dokumente/loT-election/guardian/components/adapter/adapter.c File Reference	3
3.1.1 Function Documentation	4
3.1.2 Variable Documentation	4
3.2 D:/Dokumente/loT-election/guardian/components/adapter/include/adapter.h File Reference	5
3.2.1 Function Documentation	6
3.3 adapter.h	6
3.4 D:/Dokumente/loT-election/guardian/components/adapter/util/include/serialize.h File Reference	6
3.4.1 Function Documentation	7
3.5 serialize.h	10
3.6 D:/Dokumente/loT-election/guardian/components/adapter/util/include/test_decrypt.h File Reference	11
3.6.1 Function Documentation	11
3.7 test_decrypt.h	12
3.8 D:/Dokumente/loT-election/guardian/components/adapter/util/serialize.c File Reference	12
3.8.1 Function Documentation	12
3.9 D:/Dokumente/loT-election/guardian/components/adapter/util/test_decrypt.c File Reference	16
3.9.1 Macro Definition Documentation	16
3.9.2 Function Documentation	16
3.10 D:/Dokumente/loT-election/guardian/components/model/include/model.h File Reference	17
3.10.1 Function Documentation	17
3.11 model.h	20
3.12 D:/Dokumente/loT-election/guardian/components/model/model.c File Reference	21
3.12.1 Function Documentation	21
3.13 D:/Dokumente/loT-election/guardian/components/model/util/constants.c File Reference	24
3.13.1 Variable Documentation	25
3.14 D:/Dokumente/loT-election/guardian/components/model/util/crypto_utils.c File Reference	25
3.14.1 Function Documentation	26
3.15 D:/Dokumente/loT-election/guardian/components/model/util/include/constants.h File Reference	33
3.15.1 Variable Documentation	33
3.16 constants.h	33
3.17 D:/Dokumente/loT-election/guardian/components/model/util/include/crypto_utils.h File Reference	34
3.17.1 Data Structure Documentation	35
3.17.2 Macro Definition Documentation	38

3.17.3 Function Documentation	38
3.18 <code>crypto_utils.h</code>	44
3.19 <code>D:/Dokuments/loT-election/guardian/components/model/util/include/Utils.h</code> File Reference	46
3.19.1 Function Documentation	47
3.20 <code>utils.h</code>	51
3.21 <code>D:/Dokuments/loT-election/guardian/components/model/util/Utils.c</code> File Reference	51
3.21.1 Function Documentation	52
3.22 <code>D:/Dokuments/loT-election/guardian/components/wolfssl/include/user_settings.h</code> File Reference	56
3.22.1 Macro Definition Documentation	58
3.23 <code>user_settings.h</code>	69
3.24 <code>D:/Dokuments/loT-election/guardian/main/main.c</code> File Reference	78
3.24.1 Function Documentation	78
3.25 <code>D:/Dokuments/loT-election/guardian/main/test_performance.c</code> File Reference	78
3.25.1 Macro Definition Documentation	79
3.25.2 Function Documentation	79
3.26 <code>D:/Dokuments/loT-election/guardian/main/test_performance.h</code> File Reference	79
3.26.1 Function Documentation	80
3.27 <code>test_performance.h</code>	80
3.28 <code>D:/Dokuments/loT-election/guardian/README.md</code> File Reference	81
Index	83

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:

D:/Dokumente/IoT-election/guardian/components/adapter/ adapter.c	3
D:/Dokumente/IoT-election/guardian/components/adapter/include/ adapter.h	5
D:/Dokumente/IoT-election/guardian/components/adapter/util/ serialize.c	12
D:/Dokumente/IoT-election/guardian/components/adapter/util/ test_decrypt.c	16
D:/Dokumente/IoT-election/guardian/components/adapter/util/include/ serialize.h	6
D:/Dokumente/IoT-election/guardian/components/adapter/util/include/ test_decrypt.h	11
D:/Dokumente/IoT-election/guardian/components/model/ model.c	21
D:/Dokumente/IoT-election/guardian/components/model/include/ model.h	17
D:/Dokumente/IoT-election/guardian/components/model/util/ constants.c	24
D:/Dokumente/IoT-election/guardian/components/model/util/ crypto_utils.c	25
D:/Dokumente/IoT-election/guardian/components/model/util/ utils.c	51
D:/Dokumente/IoT-election/guardian/components/model/util/include/ constants.h	33
D:/Dokumente/IoT-election/guardian/components/model/util/include/ crypto_utils.h	34
D:/Dokumente/IoT-election/guardian/components/model/util/include/ utils.h	46
D:/Dokumente/IoT-election/guardian/components/wolfssl/include/ user_settings.h	56
D:/Dokumente/IoT-election/guardian/main/ main.c	78
D:/Dokumente/IoT-election/guardian/main/ test_performance.c	78
D:/Dokumente/IoT-election/guardian/main/ test_performance.h	79

3 File Documentation

3.1 D:/Dokumente/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/loT-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.](#)

```
00001 #ifndef ADAPTER_H
00002 #define ADAPTER_H
00003
00004 #include "adapter.h"
00005 #include "model.h"
00006 #include <stdio.h>
00007 #include <stdint.h>
00008 #include <stddef.h>
00009 #include <string.h>
00010 #include "esp_wifi.h"
00011 #include "esp_system.h"
00012 #include "esp_event.h"
00013 #include "esp_netif.h"
00014 #include "protocol_examples_common.h"
00015
00016 #include "freertos/FreeRTOS.h"
00017 #include "freertos/task.h"
00018 #include "freertos/semphr.h"
00019 #include "freertos/queue.h"
00020
00021 #include "lwip/sockets.h"
00022 #include "lwip/dns.h"
00023 #include "lwip/netdb.h"
00024
00025 #include "esp_log.h"
00026 #include "mqtt_client.h"
00027 #include "esp_mac.h"
00028 #include "serialize.h"
00029 #include "test_decrypt.h"
00030
00031
00032 void mqtt_app_start(void);
00033 #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

<i>buffer</i>	Pointer to the byte array containing the serialized <code>CiphertextTally</code> data.
<i>len</i>	The length of the byte array.
<i>ciphertally</i>	Pointer to the <code>CiphertextTally</code> 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

<i>buffer</i>	Pointer to the byte array containing the serialized ElectionKeyPair data.
<i>len</i>	The length of the byte array.
<i>key_pair</i>	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

<i>buffer</i>	Pointer to the byte array containing the serialized ElectionPartialKeyPairBackup data.
<i>len</i>	The length of the byte array.
<i>backup</i>	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

<i>buffer</i>	Pointer to the byte array containing the serialized ElectionPartialKeyVerification data.
<i>len</i>	The length of the byte array.
<i>verification</i>	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

<i>share</i>	Pointer to the DecryptionShare object to serialize.
<i>len</i>	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

<i>key_pair</i>	Pointer to the ElectionKeyPair object to serialize.
<i>len</i>	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

<i>backup</i>	Pointer to the ElectionPartialKeyPairBackup object to serialize.
<i>len</i>	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

<i>verification</i>	Pointer to the ElectionPartialKeyVerification object to serialize.
<i>len</i>	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.](#)

```
00001 #ifndef SERIALIZE_H
00002 #define SERIALIZE_H
```

```

00003
00004 #include "crypto_utils.h"
00005 #include "buff.pb-c.h"
00006 #include "tally.pb-c.h"
00007 #include <wolfssl/wolfcrypt/wolfmath.h>
00008
00009
00010 uint8_t* serialize_election_partial_key_verification(ElectionPartialKeyVerification* verification,
    unsigned* len);
00011 uint8_t* serialize_election_partial_key_backup(ElectionPartialKeyPairBackup* backup, unsigned* len);
00012 uint8_t* serialize_election_key_pair(ElectionKeyPair* key_pair, unsigned* len);
00013 uint8_t* serialize_DecryptionShare(DecryptionShare* share, unsigned* len);
00014
00015 int deserialize_election_partial_key_verification(uint8_t* buffer, unsigned len,
    ElectionPartialKeyVerification* verification);
00016 int deserialize_election_partial_key_backup(uint8_t* buffer, unsigned len,
    ElectionPartialKeyPairBackup* backup);
00017 int deserialize_election_key_pair(uint8_t* buffer, unsigned len, ElectionKeyPair* key_pair);
00018 int deserialize_ciphertext_tally(uint8_t* buffer, unsigned len, CiphertextTally* ciphertextally);
00019
00020 #endif // SERIALIZE_H
00021
00022

```

3.6 D:/Dokumente/loT-election/guardian/components/adapters/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](#) *ciphertextally)
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

<i>guardian</i>	Pointer to the ElectionKeyPair object used for decryption.
<i>tally</i>	Pointer to the CiphertextTally object to be decrypted.

3.7 test_decrypt.h

Go to the documentation of this file.

```
00001 #ifndef TEST_DECRYPT_H
00002 #define TEST_DECRYPT_H
00003 #include "test_decrypt.h"
00004 #include "model.h"
00005 #include "crypto_utils.h"
00006 #include <stdio.h>
00007 #include "esp_timer.h"
00008 #include <math.h>
00009 #include "esp_task_wdt.h"
00010
00011 void perform_measurement(ElectionKeyPair *guardian, CiphertextTally *ciphertally);
00012
00013 #endif // TEST_DECRYPT_H
```

3.8 D:/Dokumente/IoT-election/guardian/components/adapters/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

<i>buffer</i>	Pointer to the byte array containing the serialized CiphertextTally data.
<i>len</i>	The length of the byte array.
<i>ciphertextally</i>	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

<i>buffer</i>	Pointer to the byte array containing the serialized ElectionKeyPair data.
<i>len</i>	The length of the byte array.
<i>key_pair</i>	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

<i>buffer</i>	Pointer to the byte array containing the serialized ElectionPartialKeyPairBackup data.
<i>len</i>	The length of the byte array.
<i>backup</i>	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

<i>buffer</i>	Pointer to the byte array containing the serialized ElectionPartialKeyVerification data.
<i>len</i>	The length of the byte array.
<i>verification</i>	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

<i>share</i>	Pointer to the DecryptionShare object to serialize.
<i>len</i>	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

<i>key_pair</i>	Pointer to the ElectionKeyPair object to serialize.
<i>len</i>	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

<i>backup</i>	Pointer to the ElectionPartialKeyPairBackup object to serialize.
<i>len</i>	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

<i>verification</i>	Pointer to the ElectionPartialKeyVerification object to serialize.
<i>len</i>	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:/Dokumente/loT-election/guardian/components/adapters/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

<i>guardian</i>	Pointer to the ElectionKeyPair object used for decryption.
<i>tally</i>	Pointer to the CiphertextTally object to be decrypted.

3.10 D:/Dokuments/loT-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

<i>guardian</i>	An ElectionKeyPair representing the current guardian.
<i>pubkey_map</i>	An array of ElectionKeyPair structures, each containing a guardian's public key.
<i>count</i>	The number of guardians (and thus the number of public keys in <code>pubkey_map</code>).
<i>joint_key</i>	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

<i>guardian</i>	The ElectionKeyPair of the guardian computing the decryption share.
<i>ciphertally</i>	A pointer to the CiphertextTally struct containing the encrypted tallies for each contest.
<i>share</i>	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

<i>quorum</i>	The number of guardians required to decrypt the election. This determines the degree of the polynomial.
<i>key_pair</i>	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

<i>sender</i>	The ElectionKeyPair of the guardian sending the backup.
<i>receiver</i>	The ElectionKeyPair of the guardian receiving the backup.
<i>backup</i>	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

<i>receiver</i>	The ElectionKeyPair of the guardian receiving and verifying the backup.
<i>sender</i>	The ElectionKeyPair of the guardian who sent the backup.
<i>backup</i>	A pointer to the ElectionPartialKeyPairBackup struct containing the encrypted backup to be verified.
<i>verification</i>	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.](#)

```
00001 #ifndef MODEL_H
00002 #define MODEL_H
00003
00004
00005 #include "utils.h"
00006 #include "crypto_utils.h"
00007 #include "model.h"
00008
00009 void test_hash();
```

```

00010 int combine_election_public_keys(ElectionKeyPair *guardian, ElectionKeyPair *pubkey_map, size_t count,
    ElectionJointKey *joint_key);
00011 int generate_election_key_pair(int quorum, ElectionKeyPair *key_pair);
00012 int generate_election_partial_key_backup(ElectionKeyPair *sender, ElectionKeyPair *receiver,
    ElectionPartialKeyPairBackup *backup);
00013 int verify_election_partial_key_backup(ElectionKeyPair *receiver, ElectionKeyPair *sender,
    ElectionPartialKeyPairBackup *backup, ElectionPartialKeyVerification *verification);
00014 int compute_decryption_share(ElectionKeyPair *guardian, CiphertextTally *ciphertally, DecryptionShare
    *share);
00015 #endif // MOD_MATH_H

```

3.12 D:/Dokuments/loT-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

<i>guardian</i>	An ElectionKeyPair representing the current guardian.
<i>pubkey_map</i>	An array of ElectionKeyPair structures, each containing a guardian's public key.
<i>count</i>	The number of guardians (and thus the number of public keys in pubkey_map).
<i>joint_key</i>	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.
Generated by Doxygen	

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

<i>guardian</i>	The ElectionKeyPair of the guardian computing the decryption share.
<i>ciphertally</i>	A pointer to the CiphertextTally struct containing the encrypted tallies for each contest.
<i>share</i>	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

<i>quorum</i>	The number of guardians required to decrypt the election. This determines the degree of the polynomial.
<i>key_pair</i>	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

<i>sender</i>	The ElectionKeyPair of the guardian sending the backup.
<i>receiver</i>	The ElectionKeyPair of the guardian receiving the backup.
<i>backup</i>	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

<i>receiver</i>	The ElectionKeyPair of the guardian receiving and verifying the backup.
<i>sender</i>	The ElectionKeyPair of the guardian who sent the backup.
<i>backup</i>	A pointer to the ElectionPartialKeyPairBackup struct containing the encrypted backup to be verified.
<i>verification</i>	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:/Dokumente/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

<i>guardian</i>	A pointer to the ElectionKeyPair structure containing the guardian's key pair.
<i>contest</i>	A pointer to the CiphertextTallyContest structure representing the contest data.
<i>base_hash</i>	A pointer to the base hash (sp_int) used in the decryption process.
<i>dec_contest</i>	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

<i>exponent_modifier</i>	Unique modifier (guardian id) for exponent [0, Q]
<i>polynomial</i>	Election polynomial
<i>coordinate</i>	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

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

<i>polynomial</i>	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 <code>num_coefficients</code> field before calling this function. This function will allocate memory for the <code>coefficients</code> array within the ElectionPolynomial struct, as well as for the <code>value</code> , <code>commitment</code> , and <code>proof</code> 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

<i>a</i>	First element
<i>b</i>	Second element
<i>result</i>	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

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

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

<i>message</i>	The message (sp_int) to be encrypted.
<i>nonce</i>	A random nonce (sp_int) used for encryption.
<i>public_key</i>	The recipient's public key (sp_int).
<i>encryption_seed</i>	An encryption seed (sp_int) used in the key derivation function.
<i>encrypted_message</i>	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

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

Parameters

<i>hash_header</i>	A pointer to a header value used when generating the challenge, typically the election extended base hash.
<i>proof</i>	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 <code>pad</code> , <code>data</code> , <code>challenge</code> , and <code>response</code> 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

<i>array</i>	The byte array to print.
<i>size</i>	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

<i>num</i>	The <code>sp_int</code> to print.
------------	-----------------------------------

rand_q()

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

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

Parameters

<i>result</i>	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

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

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

Returns

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

3.15 D:/Dokuments/loT-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.](#)

```
00001 #ifndef CONSTANTS_H
00002 #define CONSTANTS_H
00003
00004 // The constants for mathematical functions during the election
00005
00006 extern const unsigned char p_3072[384];
00007 extern const unsigned char g_3072[384];
00008 extern const unsigned char q_256[32];
00009
00010 #endif // CONSTANTS_H
```

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

```
#include <wolfssl/wolfcrypt/port/Esspressif/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

<code>CiphertextDecryptionContest *</code>	<code>contests</code>	
<code>uint8_t</code>	<code>guardian_id[6]</code>	
<code>int</code>	<code>num_contest</code>	
<code>char *</code>	<code>object_id</code>	
<code>sp_int *</code>	<code>public_key</code>	

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

<i>guardian</i>	A pointer to the <code>ElectionKeyPair</code> structure containing the guardian's key pair.
<i>contest</i>	A pointer to the <code>CiphertextTallyContest</code> structure representing the contest data.
<i>base_hash</i>	A pointer to the base hash (<code>sp_int</code>) used in the decryption process.
<i>dec_contest</i>	A pointer to the <code>CiphertextDecryptionContest</code> 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

<i>exponent_modifier</i>	Unique modifier (guardian id) for exponent [0, Q]
<i>polynomial</i>	Election polynomial
<i>coordinate</i>	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

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

<i>polynomial</i>	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 <code>num_coefficients</code> field before calling this function. This function will allocate memory for the <code>coefficients</code> array within the ElectionPolynomial struct, as well as for the <code>value</code> , <code>commitment</code> , and <code>proof</code> 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

<i>a</i>	First element
<i>b</i>	Second element
<i>result</i>	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

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

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

<i>message</i>	The message (sp_int) to be encrypted.
<i>nonce</i>	A random nonce (sp_int) used for encryption.
<i>public_key</i>	The recipient's public key (sp_int).
<i>encryption_seed</i>	An encryption seed (sp_int) used in the key derivation function.
<i>encrypted_message</i>	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

<i>alpha</i>	A pointer to the ElGamal ciphertext's alpha component ($(g^r \bmod p)$), which is a large integer.
--------------	--

Parameters

<i>beta</i>	A pointer to the ElGamal ciphertext's beta component ($(m \cdot y^r \bmod p)$), which is a large integer.
<i>secret</i>	A pointer to the nonce ((r)) used to encrypt the message, which is a large integer.
<i>m</i>	A pointer to the plaintext message ((m)), which is a large integer.
<i>seed</i>	A pointer to a seed value used to generate random values within the proof.
<i>hash_header</i>	A pointer to a header value used when generating the challenge, typically the election extended base hash.
<i>proof</i>	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 <code>pad</code> , <code>data</code> , <code>challenge</code> , and <code>response</code> 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

<i>array</i>	The byte array to print.
<i>size</i>	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

<i>num</i>	The sp_int to print.
------------	----------------------

rand_q()

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

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

Parameters

<i>result</i>	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

<i>exponent_modifier</i>	The unique identifier of the guardian (usually sequence order).
<i>polynomial</i>	The election polynomial to verify against.
<i>coordinate</i>	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.](#)

```
00001 #ifndef CRYPTO_UTILS_H
00002 #define CRYPTO_UTILS_H
00003
00004 #include <wolfssl/wolfcrypt/port/Espressif/esp32-crypt.h>
00005 #include <wolfssl/wolfcrypt/wolfmath.h>
```

```

00006 #include <wolfssl/ssl.h>
00007 #include "esp_log.h"
00008 #include "esp_random.h"
00009 #include "constants.h"
00010
00011
00012
00013 #define BLOCK_SIZE 32
00014 #define MODIFIED_CHUNK_SIZE (BLOCK_SIZE + 8)
00015
00016 typedef struct {
00017     sp_int* pubkey;
00018     sp_int* commitment;
00019     sp_int* challenge;
00020     sp_int* response;
00021 } SchnorrProof;
00022
00023 typedef struct {
00024     sp_int* value;
00025     sp_int* commitment;
00026     SchnorrProof proof;
00027 } Coefficient;
00028
00029 typedef struct {
00030     int num_coefficients;
00031     Coefficient* coefficients;
00032 } ElectionPolynomial;
00033
00034 // contains also private key. Be careful when sending!
00035 typedef struct {
00036     uint8_t guardian_id[6];
00037     sp_int* public_key;
00038     sp_int* private_key;
00039     ElectionPolynomial polynomial;
00040 } ElectionKeyPair;
00041
00042 typedef struct {
00043     sp_int* pad;
00044     sp_int* data;
00045     sp_int* mac;
00046 } HashedElGamalCiphertext;
00047
00048 typedef struct {
00049     uint8_t sender[6];
00050     uint8_t receiver[6];
00051     HashedElGamalCiphertext encrypted_coordinate;
00052 } ElectionPartialKeyPairBackup;
00053
00054
00055 typedef struct {
00056     uint8_t sender[6];
00057     uint8_t receiver[6];
00058     uint8_t verifier[6];
00059     bool verified;
00060 } ElectionPartialKeyVerification;
00061
00062 typedef struct {
00063     sp_int* joint_key;
00064     sp_int* commitment_hash;
00065 } ElectionJointKey;
00066
00067 typedef struct {
00068     char* object_id;
00069     sp_int* ciphertext_pad;
00070     sp_int* ciphertext_data;
00071 } CiphertextTallySelection;
00072
00073 typedef struct {
00074     char* object_id;
00075     int sequence_order;
00076     sp_int* description_hash;
00077     int num_selections;
00078     CiphertextTallySelection* selections;
00079 } CiphertextTallyContest;
00080
00081 typedef struct {
00082     char* object_id;
00083     sp_int* base_hash;
00084     int num_contest;
00085     CiphertextTallyContest* contests;
00086 } CiphertextTally;
00087
00088 typedef struct {
00089     sp_int* pad;
00090     sp_int* data;
00091     sp_int* challenge;
00092     sp_int* response;

```

```

00093 } ChaumPedersenProof;
00094
00095 typedef struct {
00096     char* object_id;
00097     uint8_t guardian_id[6];
00098     sp_int* decryption;
00099     ChaumPedersenProof proof;
00100 } CiphertextDecryptionSelection;
00101
00102 typedef struct{
00103     char* object_id;
00104     uint8_t guardian_id[6];
00105     sp_int *public_key;
00106     sp_int* description_hash;
00107     int num_selections;
00108     CiphertextDecryptionSelection* selections;
00109 } CiphertextDecryptionContest;
00110
00111 typedef struct {
00112     char* object_id;
00113     uint8_t guardian_id[6];
00114     sp_int* public_key;
00115     int num_contest;
00116     CiphertextDecryptionContest* contests;
00117 } DecryptionShare;
00118
00119 int elgamal_combine_public_keys(ElectionKeyPair *guardian, ElectionKeyPair *pubkey_map, size_t count,
    sp_int *key);
00120 int compute_decryption_share_for_contest(ElectionKeyPair *guardian, CiphertextTallyContest *contest,
    sp_int* base_hash , CiphertextDecryptionContest *dec_contest);
00121
00122 int compute_polynomial_coordinate(uint8_t* exponent_modifier, ElectionPolynomial* polynomial, sp_int
    *coordinate);
00123 int verify_polynomial_coordinate(uint8_t* exponent_modifier, ElectionPolynomial* polynomial, sp_int
    *coordinate);
00124
00125 int hashed_elgamal_encrypt(sp_int *message, sp_int *nonce, sp_int *public_key, sp_int
    *encryption_seed, HashedElGamalCiphertext *encrypted_message);
00126 int hashed_elgamal_decrypt(HashedElGamalCiphertext *encrypted_message, sp_int *secret_key, sp_int
    *encryption_seed, sp_int *message);
00127
00128 int generate_polynomial(ElectionPolynomial *polynomial);
00129
00130 int hash(sp_int *a, sp_int *b, sp_int *result);
00131 int hash_keys(ElectionKeyPair *guardian, ElectionKeyPair *pubkey_map, size_t count, sp_int
    *commitment);
00132 int rand_q(sp_int *result);
00133 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);
00134
00135 void print_sp_int(sp_int *num);
00136 void print_byte_array(const byte *array, int size);
00137
00138 #endif // CRYPTO_UTILS_H

```

3.19 D:/Dokumente/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

<i>proof</i>	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_CiphertextDecryptionSelection](#) for each element before freeing the array itself. It also sets pointers to NULL to prevent double freeing.

Parameters

<i>contest</i>	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

<i>selection</i>	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

<i>tally</i>	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

<i>contest</i>	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

<i>selection</i>	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

<i>coefficient</i>	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

<i>share</i>	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

<i>key_pair</i>	A pointer to the ElectionKeyPair struct to free. If <code>key_pair</code> 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

<i>backup</i>	A pointer to the ElectionPartialKeyPairBackup struct to free. If <code>backup</code> 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

<i>polynomial</i>	A pointer to the ElectionPolynomial struct to free. If <code>polynomial</code> 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

<i>proof</i>	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.](#)

```
00001 #ifndef UTILS_H
00002 #define UTILS_H
00003
00004 #include <stdint.h>
00005 #include <wolfssl/wolfcrypt/port/Espressif/esp32-crypt.h>
00006 #include <wolfssl/wolfcrypt/wolfmath.h>
00007 #include <wolfssl/ssl.h>
00008 #include "crypto_utils.h"
00009
00010 void free_DecryptionShare(DecryptionShare* share);
00011 void free_CiphertextTally(CiphertextTally* tally);
00012 void free_CiphertextTallySelection(CiphertextTallySelection* selection);
00013 void free_CiphertextTallyContest(CiphertextTallyContest* contest);
00014 void free_ChaumPedersenProof(ChaumPedersenProof* proof);
00015 void free_CiphertextDecryptionSelection(CiphertextDecryptionSelection* selection);
00016 void free_CiphertextDecryptionContest(CiphertextDecryptionContest* contest);
00017 void free_ElectionPartialKeyPairBackup(ElectionPartialKeyPairBackup* backup);
00018 void free_ElectionKeyPair(ElectionKeyPair *key_pair);
00019 void free_ElectionPolynomial(ElectionPolynomial *polynomial);
00020 void free_Coefficient(Coefficient *coefficient);
00021 void free_SchnorrProof(SchnorrProof *proof);
00022
00023 #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

<i>proof</i>	A pointer to the <code>ChaumPedersenProof</code> 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_CiphertextDecryptionSelection` for each element before freeing the array itself. It also sets pointers to NULL to prevent double freeing.

Parameters

<i>contest</i>	A pointer to the <code>CiphertextDecryptionContest</code> 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_ChauPedersenProof to free the proof member. It sets pointers to NULL to prevent double freeing.

Parameters

<i>selection</i>	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

<i>tally</i>	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

<i>contest</i>	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

<i>selection</i>	A pointer to the CiphertextTallySelection object to free. If <code>selection</code> 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

<i>coefficient</i>	A pointer to the Coefficient struct to free. If <code>coefficient</code> 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

<i>share</i>	A pointer to the DecryptionShare struct to free. If <code>share</code> 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

<i>key_pair</i>	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

<i>backup</i>	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

<i>polynomial</i>	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

<i>proof</i>	A pointer to the SchnorrProof struct to free. If proof is NULL, the function returns immediately.
--------------	---

3.22 D:/Dokumente/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 OPENSSSL_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
```

SCRIPT_TEST_ALL

```
#define SCRIPT_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.](#)

```
00001 /* user_settings.h
00002  *
00003  * Copyright (C) 2006-2024 wolfSSL Inc.
00004  *
00005  * This file is part of wolfSSL.
00006  *
00007  * wolfSSL is free software; you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License as published by
00009  * the Free Software Foundation; either version 2 of the License, or
00010  * (at your option) any later version.
00011  *
00012  * wolfSSL is distributed in the hope that it will be useful,
00013  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00015  * GNU General Public License for more details.
00016  *
00017  * You should have received a copy of the GNU General Public License
00018  * along with this program; if not, write to the Free Software
00019  * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1335, USA
00020  */
00021
00022 /* This user_settings.h is for Espressif ESP-IDF
00023  *
00024  * Standardized wolfSSL Espressif ESP32 + ESP8266 user_settings.h V5.7.0-1
00025  *
00026  * Do not include any wolfssl headers here
00027  *
00028  * When editing this file:
00029  * ensure wolfssl_test and wolfssl_benchmark settings match.
00030  */
00031
00032 /* The Espressif project config file. See also sdkconfig.defaults */
00033 #include "sdkconfig.h"
00034
00035 /* The Espressif sdkconfig will have chipset info.
00036 **
00037 ** Some possible values:
00038 **
00039 ** CONFIG_IDF_TARGET_ESP32
00040 ** CONFIG_IDF_TARGET_ESP32S2
00041 ** CONFIG_IDF_TARGET_ESP32S3
00042 ** CONFIG_IDF_TARGET_ESP32C3
00043 ** CONFIG_IDF_TARGET_ESP32C6
00044 */
00045
00046 #undef WOLFSSL_ESPIDF
00047 #define WOLFSSL_ESPIDF
00048
00049 /* We don't use WiFi, so don't compile in the esp-sdk-lib WiFi helpers: */
00050 #define NO_ESP_SDK_WIFI
00051
00052 /* Experimental Kyber */
00053 #if 0
00054 /* Kyber typically needs a minimum 10K stack */
00055 #define WOLFSSL_EXPERIMENTAL_SETTINGS
00056 #define WOLFSSL_HAVE_KYBER
00057 #define WOLFSSL_WC_KYBER
00058 #define WOLFSSL_SHA3
00059 #endif
00060
00061 /*
00062  * ONE of these Espressif chip families will be detected from sdkconfig:
00063  *
00064  * WOLFSSL_ESP32
00065  * WOLFSSL_ESP8266

```

```

00066 */
00067 #undef WOLFSSL_ESPROOM32SE
00068 #undef WOLFSSL_ESP8266
00069 #undef WOLFSSL_ESP32
00070 /* See below for chipset detection from sdkconfig.h */
00071
00072 /* when you want to use SINGLE THREAD. Note Default ESP-IDF is FreeRTOS */
00073 /* #define SINGLE_THREADED */
00074
00075 /* SMALL_SESSION_CACHE saves a lot of RAM for ClientCache and SessionCache.
00076  * Memory requirement is about 5KB, otherwise 20K is needed when not specified.
00077  * If extra small footprint is needed, try MICRO_SESSION_CACHE (< 1K)
00078  * When really desperate or no TLS used, try NO_SESSION_CACHE. */
00079 #define NO_SESSION_CACHE
00080
00081 /* Small Stack uses more heap. */
00082 #define WOLFSSL_SMALL_STACK
00083
00084 /* Full debugging turned off, but show malloc failure detail */
00085 /* #define DEBUG_WOLFSSL */
00086 #define DEBUG_WOLFSSL_MALLOC
00087
00088 /* See test.c that sets cert buffers; we'll set them here: */
00089 #define USE_CERT_BUFFERS_256
00090 #define USE_CERT_BUFFERS_2048
00091
00092 /* RSA_LOW_MEM: Half as much memory but twice as slow. */
00093 #define RSA_LOW_MEM
00094
00095 // * FP_MAX_BITS defaults to 4096 bits. This allows only multiplication of up to 2048x2048 bits.
00096 #define FP_MAX_BITS 6144
00097
00098 /* Uncommon settings for testing only */
00099 #define TEST_ESPIDF_ALL_WOLFSSL
00100 #ifndef TEST_ESPIDF_ALL_WOLFSSL
00101     #define WOLFSSL_MD2
00102     #define HAVE_BLAKE2
00103     #define HAVE_BLAKE2B
00104     #define HAVE_BLAKE2S
00105
00106     #define WC_RC2
00107     #define WOLFSSL_ALLOW_RC4
00108
00109     #define HAVE_POLY1305
00110
00111     #define WOLFSSL_AES_128
00112     #define WOLFSSL_AES_OFB
00113     #define WOLFSSL_AES_CFB
00114     #define WOLFSSL_AES_XTS
00115
00116     /* #define WC_SRTTP_KDF */
00117     /* TODO Causes failure with Espressif AES HW Enabled */
00118     /* #define HAVE_AES_ECB */
00119     /* #define HAVE_AESCCM */
00120     /* TODO sanity check when missing HAVE_AES_ECB */
00121     #define WOLFSSL_WOLFSSH
00122
00123     #define HAVE_AESGCM
00124     #define WOLFSSL_AES_COUNTER
00125
00126     #define HAVE_FFDHE
00127     #define HAVE_FFDHE_2048
00128     #if defined(CONFIG_IDF_TARGET_ESP8266)
00129         /* TODO Full size SRP is disabled on the ESP8266 at this time.
00130          * Low memory issue? */
00131         #define WOLFCRYPT_HAVE_SRP
00132         /* MIN_FFDHE_FP_MAX_BITS = (MIN_FFDHE_BITS * 2); see settings.h */
00133         #define FP_MAX_BITS MIN_FFDHE_FP_MAX_BITS
00134     #elif defined(CONFIG_IDF_TARGET_ESP32) || \
00135           defined(CONFIG_IDF_TARGET_ESP32S2) || \
00136           defined(CONFIG_IDF_TARGET_ESP32S3)
00137         #define WOLFCRYPT_HAVE_SRP
00138         #define FP_MAX_BITS (8192 * 2)
00139     #elif defined(CONFIG_IDF_TARGET_ESP32C3) || \
00140           defined(CONFIG_IDF_TARGET_ESP32H2)
00141         /* SRP Known to be working on this target: */
00142         #define WOLFCRYPT_HAVE_SRP
00143         #define FP_MAX_BITS (8192 * 2)
00144     #else
00145         /* For everything else, give a try and see if SRP working: */
00146         #define WOLFCRYPT_HAVE_SRP
00147         #define FP_MAX_BITS (8192 * 2)
00148     #endif
00149
00150     #define HAVE_DH
00151
00152     /* TODO: there may be a problem with HAVE_CAMELLIA with HW AES disabled.

```



```

00153      * Do not define NO_WOLFSSL_ESP32_CRYPT_AES when enabled: */
00154      /* #define HAVE_CAMELLIA */
00155
00156      /* DSA requires old SHA */
00157      #define HAVE_DSA
00158
00159      /* Needs SHA512 ? */
00160      #define HAVE_HPKE
00161
00162      /* Not for Espressif? */
00163      #if defined(CONFIG_IDF_TARGET_ESP32C2) || \
00164          defined(CONFIG_IDF_TARGET_ESP8684) || \
00165          defined(CONFIG_IDF_TARGET_ESP32H2) || \
00166          defined(CONFIG_IDF_TARGET_ESP8266)
00167
00168          #if defined(CONFIG_IDF_TARGET_ESP8266)
00169              #undef HAVE_ECC
00170              #undef HAVE_ECC_CDH
00171              #undef HAVE_CURVE25519
00172
00173              /* TODO does CHACHA also need alignment? Failing on ESP8266
00174               * See SHA256 __attribute__((aligned(4))); and WC_SHA256_ALIGN */
00175              #ifdef HAVE_CHACHA
00176                  #error "HAVE_CHACHA not supported on ESP8266"
00177              #endif
00178              #ifdef HAVE_XCHACHA
00179                  #error "HAVE_XCHACHA not supported on ESP8266"
00180              #endif
00181          #else
00182              #define HAVE_XCHACHA
00183              #define HAVE_CHACHA
00184              /* TODO Not enabled at this time, needs further testing:
00185               * #define WC_SRTTP_KDF
00186               * #define HAVE_COMP_KEY
00187               * #define WOLFSSL_HAVE_XMSS
00188               */
00189          #endif
00190          /* TODO AES-EAX not working on this platform */
00191
00192          /* Optionally disable DH
00193           * #undef HAVE_DH
00194           * #undef HAVE_FFDHE
00195           */
00196
00197          /* ECC_SHAMIR out of memory on ESP32-C2 during ECC */
00198          #ifndef HAVE_ECC
00199              #define ECC_SHAMIR
00200          #endif
00201      #else
00202          #define WOLFSSL_AES_EAX
00203
00204          #define ECC_SHAMIR
00205      #endif
00206
00207      /* Only for WOLFSSL_IMX6_CAAM / WOLFSSL_QNX_CAAM ? */
00208      /* #define WOLFSSL_CAAM */
00209      /* #define WOLFSSL_CAAM_BLOB */
00210
00211      #define WOLFSSL_AES_SIV
00212      #define WOLFSSL_CMAC
00213
00214      #define WOLFSSL_CERT_PIV
00215
00216      /* HAVE_SCRYPT may turn on HAVE_PBKDF2 see settings.h */
00217      /* #define HAVE_SCRYPT */
00218      #define SCRYPT_TEST_ALL
00219      #define HAVE_X963_KDF
00220  #endif
00221
00222  /* optionally turn off SHA512/224 SHA512/256 */
00223  /* #define WOLFSSL_NOSHA512_224 */
00224  /* #define WOLFSSL_NOSHA512_256 */
00225  // Wolfssl was deprecated in ESP-TLS.
00226  //FREERTOS_ENABLE_BACKWARD_COMPATIBILITY
00227
00228  /* when you want to use SINGLE THREAD. Note Default ESP-IDF is FreeRTOS */
00229  /* #define SINGLE_THREADED */
00230
00231  /* Turning off features reducing library size */
00232  #define NO_SHA
00233  #define NO_OLD_TLS
00234  #define NO_DSA
00235  #define NO_MD4
00236  #define NO_MD5
00237  #define NO_PWDBASED
00238  #define NO_RC4
00239  #define WOLFSSL_SP_NO_256

```

```

00240 #define NO_CRYPT_BENCHMARK
00241 #define NO_CRYPT_TEST
00242 #define NO_PSK
00243 #define NO_TLS
00244
00245 // Hardening Enable TFM Timing Resistant Code
00246 #define TFM_TIMING_RESISTANT
00247
00248 #define BENCH_EMBEDDED
00249
00250 /* TLS 1.3 */
00251 #define WOLFSSL_TLS13
00252 #define HAVE_TLS_EXTENSIONS
00253 #define WC_RSA_PSS
00254 #define HAVE_HKDF
00255 #define HAVE_AEAD
00256 #define HAVE_SUPPORTED_CURVES
00257 #define WC_KDF_NIST_SP_800_56C
00258
00259 #define WOLFSSL_BENCHMARK_FIXED_UNITS_KB
00260
00261 #define NO_FILESYSTEM
00262
00263 #define NO_OLD_TLS
00264
00265 #define HAVE_AESGCM
00266
00267 /* Optional RIPEMD: RACE Integrity Primitives Evaluation Message Digest */
00268 /* #define WOLFSSL_RIPEMD */
00269
00270 /* when you want to use SHA224 */
00271 // #define WOLFSSL_SHA224 ++
00272
00273 /* when you want to use SHA384 */
00274 // #define WOLFSSL_SHA384 ++
00275
00276 /* when you want to use SHA512 */
00277 #define WOLFSSL_SHA512
00278
00279 /* when you want to use SHA3 */
00280 // #define WOLFSSL_SHA3 ++
00281
00282 /* ED25519 requires SHA512 */
00283 // #define HAVE_ED25519 ++
00284
00285 /* Some features not enabled for ESP8266: */
00286 #if defined(CONFIG_IDF_TARGET_ESP8266) || \
00287     defined(CONFIG_IDF_TARGET_ESP32C2)
00288     /* TODO determine low memory configuration for ECC. */
00289 #else
00290     #define HAVE_ECC
00291     #define HAVE_CURVE25519
00292     #define CURVE25519_SMALL
00293 #endif
00294
00295 #define HAVE_ED25519
00296
00297 /* Optional OPENSSL compatibility */
00298 #define OPENSSL_EXTRA
00299
00300 /* #Optional HAVE_PKCS7 */
00301 // #define HAVE_PKCS7 ++
00302
00303 #if defined(HAVE_PKCS7)
00304     /* HAVE_PKCS7 may enable HAVE_PBKDF2 see settings.h */
00305     #define NO_PBKDF2
00306
00307     #define HAVE_AES_KEYWRAP
00308     #define HAVE_X963_KDF
00309     #define WOLFSSL_AES_DIRECT
00310 #endif
00311
00312 /* when you want to use AES counter mode */
00313 /* #define WOLFSSL_AES_DIRECT */
00314 /* #define WOLFSSL_AES_COUNTER */
00315
00316 /* esp32-wroom-32se specific definition */
00317 #if defined(WOLFSSL_ESPROOM32SE)
00318     #define WOLFSSL_ATECC508A
00319     #define HAVE_PK_CALLBACKS
00320     /* when you want to use a custom slot allocation for ATECC508A */
00321     /* unless your configuration is unusual, you can use default */
00322     /* implementation. */
00323     /* #define CUSTOM_SLOT_ALLOCATION */
00324 #endif
00325
00326 /* WC_NO_CACHE_RESISTANT: slower but more secure */

```

```

00327 /* #define WC_NO_CACHE_RESISTANT */
00328
00329 /* TFM_TIMING_RESISTANT: slower but more secure */
00330 /* #define TFM_TIMING_RESISTANT */
00331
00332 /* #define WOLFSSL_ATECC508A_DEBUG */
00333
00334 /* date/time */
00335 /* if it cannot adjust time in the device, */
00336 /* enable macro below */
00337 /* #define NO_ASN_TIME */
00338 /* #define XTIME time */
00339
00340
00341 /* adjust wait-timeout count if you see timeout in RSA HW acceleration */
00342 #define ESP_RSA_TIMEOUT_CNT 0x349F00
00343
00344 /* hash limit for test.c */
00345 #define HASH_SIZE_LIMIT
00346
00347 /* USE_FAST_MATH is default */
00348 #define USE_FAST_MATH
00349
00350 /***** Use SP_MATH *****/
00351 #undef USE_FAST_MATH //+-
00352 #define SP_MATH //+-
00353 #define WOLFSSL_SP_MATH_ALL //+-
00354
00355 #define WOLFSSL_SP_SMALL //++
00356 #define WOLFSSL_SP_LOW_MEM //++
00357 #define WOLFSSL_HAVE_SP_RSA //++
00358 #define WOLFSSL_SP_4096 //++
00359 /* #define WOLFSSL_SP_RISCV32 */
00360
00361 /***** Use Integer Heap Math *****/
00362 /* #undef USE_FAST_MATH */
00363 /* #define USE_INTEGER_HEAP_MATH */
00364
00365
00366 #define WOLFSSL_SMALL_STACK
00367
00368
00369 #define HAVE_VERSION_EXTENDED_INFO
00370 /* #define HAVE_WC_INTROSPECTION */
00371
00372 #define HAVE_SESSION_TICKET
00373
00374 /* #define HAVE_HASHDRBG */
00375
00376 #define WOLFSSL_KEY_GEN
00377 #define WOLFSSL_CERT_REQ
00378 #define WOLFSSL_CERT_GEN
00379 #define WOLFSSL_CERT_EXT
00380 #define WOLFSSL_SYS_CA_CERTS
00381
00382
00383 #define WOLFSSL_CERT_TEXT
00384
00385 #define WOLFSSL_ASN_TEMPLATE
00386
00387 /*
00388 #undef WOLFSSL_KEY_GEN
00389 #undef WOLFSSL_CERT_REQ
00390 #undef WOLFSSL_CERT_GEN
00391 #undef WOLFSSL_CERT_EXT
00392 #undef WOLFSSL_SYS_CA_CERTS
00393 */
00394
00395 /* command-line options
00396 --enable-keygen
00397 --enable-certgen
00398 --enable-certreq
00399 --enable-certtext
00400 --enable-asn-template
00401 */
00402
00403 /* Chipset detection from sdkconfig.h
00404 * Default is HW enabled unless turned off.
00405 * Uncomment lines to force SW instead of HW acceleration */
00406 #if defined(CONFIG_IDF_TARGET_ESP32)
00407     #define WOLFSSL_ESP32
00408     /* Alternatively, if there's an ECC Secure Element present: */
00409     /* #define WOLFSSL_ESPWROOM32SE */
00410
00411     /* wolfSSL HW Acceleration supported on ESP32. Uncomment to disable: */
00412     /* #define NO_ESP32_CRYPT */
00413     /* #define NO_WOLFSSL_ESP32_CRYPT_HASH */

```

```

00414      /* #define NO_WOLFSSL_ESP32_CRYPT_AES */
00415      /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI */
00416      /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL */
00417      /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD */
00418      /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */
00419
00420      /* These are defined automatically in esp32-crypt.h, here for clarity: */
00421      #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA224 /* no SHA224 HW on ESP32 */
00422
00423      #undef ESP_RSA_MULM_BITS
00424      #define ESP_RSA_MULM_BITS 16 /* TODO add compile-time warning */
00425      /***** END CONFIG_IDF_TARGET_ESP32 *****/
00426
00427      #elif defined(CONFIG_IDF_TARGET_ESP32S2)
00428          #define WOLFSSL_ESP32
00429          /* wolfSSL HW Acceleration supported on ESP32-S2. Uncomment to disable: */
00430          /* #define NO_ESP32_CRYPT */
00431          /* #define NO_WOLFSSL_ESP32_CRYPT_HASH */
00432          /* Note: There's no AES192 HW on the ESP32-S2; falls back to SW */
00433          /* #define NO_WOLFSSL_ESP32_CRYPT_AES */
00434          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI */
00435          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL */
00436          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD */
00437          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */
00438          /***** END CONFIG_IDF_TARGET_ESP32S2 *****/
00439
00440      #elif defined(CONFIG_IDF_TARGET_ESP32S3)
00441          #define WOLFSSL_ESP32
00442          /* wolfSSL HW Acceleration supported on ESP32-S3. Uncomment to disable: */
00443          /* #define NO_ESP32_CRYPT */
00444          /* #define NO_WOLFSSL_ESP32_CRYPT_HASH */
00445          /* Note: There's no AES192 HW on the ESP32-S3; falls back to SW */
00446          /* #define NO_WOLFSSL_ESP32_CRYPT_AES */
00447          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI */
00448          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL */
00449          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD */
00450          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */
00451          /***** END CONFIG_IDF_TARGET_ESP32S3 *****/
00452
00453      #elif defined(CONFIG_IDF_TARGET_ESP32C2) || \
00454          defined(CONFIG_IDF_TARGET_ESP8684)
00455          #define WOLFSSL_ESP32
00456          /* ESP8684 is essentially ESP32-C2 chip + flash embedded together in a
00457           * single QFN 4x4 mm package. Out of released documentation, Technical
00458           * Reference Manual as well as ESP-IDF Programming Guide is applicable
00459           * to both ESP32-C2 and ESP8684.
00460           *
00461           * See:
00462           * https://www.esp32.com/viewtopic.php?f=5&t=27926#:~:text=ESP8684%20is%20essentially%20ESP32%2DC2,both%20ESP32%2DC2%20and
00463           */
00464          /* wolfSSL HW Acceleration supported on ESP32-C2. Uncomment to disable: */
00465          /* #define NO_ESP32_CRYPT */
00466          /* #define NO_WOLFSSL_ESP32_CRYPT_HASH */ /* to disable all SHA HW */
00467
00468          /* These are defined automatically in esp32-crypt.h, here for clarity */
00469          #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA384 /* no SHA384 HW on C2 */
00470          #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA512 /* no SHA512 HW on C2 */
00471
00472          /* There's no AES or RSA/Math accelerator on the ESP32-C2
00473           * Auto defined with NO_WOLFSSL_ESP32_CRYPT_RSA_PRI, for clarity: */
00474          #define NO_WOLFSSL_ESP32_CRYPT_AES
00475          #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
00476          #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL
00477          #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD
00478          #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD
00479          /***** END CONFIG_IDF_TARGET_ESP32C2 *****/
00480
00481      #elif defined(CONFIG_IDF_TARGET_ESP32C3)
00482          #define WOLFSSL_ESP32
00483          /* wolfSSL HW Acceleration supported on ESP32-C3. Uncomment to disable: */
00484          /* #define NO_ESP32_CRYPT */
00485          /* #define NO_WOLFSSL_ESP32_CRYPT_HASH */ /* to disable all SHA HW */
00486
00487          /* These are defined automatically in esp32-crypt.h, here for clarity: */
00488          #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA384 /* no SHA384 HW on C6 */
00489          #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA512 /* no SHA512 HW on C6 */
00490
00491          /* #define NO_WOLFSSL_ESP32_CRYPT_AES */
00492          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI */
00493          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL */
00494          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD */
00495          /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */
00496          /***** END CONFIG_IDF_TARGET_ESP32C3 *****/
00497
00498      #elif defined(CONFIG_IDF_TARGET_ESP32C6)

```

```

00499     #define WOLFSSL_ESP32
00500     /* wolfSSL HW Acceleration supported on ESP32-C6. Uncomment to disable: */
00501
00502     /* #define NO_ESP32_CRYPT */
00503     /* #define NO_WOLFSSL_ESP32_CRYPT_HASH */
00504     /* These are defined automatically in esp32-crypt.h, here for clarity: */
00505     #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA384 /* no SHA384 HW on C6 */
00506     #define NO_WOLFSSL_ESP32_CRYPT_HASH_SHA512 /* no SHA512 HW on C6 */
00507
00508     /* #define NO_WOLFSSL_ESP32_CRYPT_AES */
00509     /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI */
00510     /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL */
00511     /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD */
00512     /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */
00513     /***** END CONFIG_IDF_TARGET_ESP32C6 *****/
00514
00515 #elif defined(CONFIG_IDF_TARGET_ESP32H2)
00516     #define WOLFSSL_ESP32
00517     /* wolfSSL Hardware Acceleration not yet implemented */
00518     #define NO_ESP32_CRYPT
00519     #define NO_WOLFSSL_ESP32_CRYPT_HASH
00520     #define NO_WOLFSSL_ESP32_CRYPT_AES
00521     #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
00522     /***** END CONFIG_IDF_TARGET_ESP32H2 *****/
00523
00524 #elif defined(CONFIG_IDF_TARGET_ESP8266)
00525     #define WOLFSSL_ESP8266
00526
00527     /* There's no hardware encryption on the ESP8266 */
00528     /* Consider using the ESP32-C2/C3/C6
00529      * See https://www.espressif.com/en/products/socs/esp32-c2 */
00530     #define NO_ESP32_CRYPT
00531     #define NO_WOLFSSL_ESP32_CRYPT_HASH
00532     #define NO_WOLFSSL_ESP32_CRYPT_AES
00533     #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
00534     /***** END CONFIG_IDF_TARGET_ESP266 *****/
00535
00536 #elif defined(CONFIG_IDF_TARGET_ESP8684)
00537     /* There's no Hardware Acceleration available on ESP8684 */
00538     #define NO_ESP32_CRYPT
00539     #define NO_WOLFSSL_ESP32_CRYPT_HASH
00540     #define NO_WOLFSSL_ESP32_CRYPT_AES
00541     #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
00542     /***** END CONFIG_IDF_TARGET_ESP8684 *****/
00543
00544 #else
00545     /* Anything else encountered, disable HW acceleration */
00546     #warning "Unexpected CONFIG_IDF_TARGET_NN value"
00547     #define NO_ESP32_CRYPT
00548     #define NO_WOLFSSL_ESP32_CRYPT_HASH
00549     #define NO_WOLFSSL_ESP32_CRYPT_AES
00550     #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
00551 #endif /* CONFIG_IDF_TARGET Check */
00552
00553 /* RSA primitive specific definition, listed AFTER the Chipset detection */
00554 #if defined(WOLFSSL_ESP32) || defined(WOLFSSL_ESPWROOM32SE)
00555     /* Consider USE_FAST_MATH and SMALL_STACK */
00556
00557     #ifndef NO_RSA
00558         #define ESP32_USE_RSA_PRIMITIVE
00559
00560         #if defined(CONFIG_IDF_TARGET_ESP32)
00561             #ifdef CONFIG_ESP_MAIN_TASK_STACK_SIZE
00562                 #if CONFIG_ESP_MAIN_TASK_STACK_SIZE < 10500
00563                     #warning "RSA may be difficult with less than 10KB Stack"
00564                 #endif
00565             #endif
00566
00567             /* NOTE HW unreliable for small values! */
00568             /* threshold for performance adjustment for HW primitive use */
00569             /* X bits of G^X mod P greater than */
00570             #undef ESP_RSA_EXPT_XBITS
00571             #define ESP_RSA_EXPT_XBITS 32
00572
00573             /* X and Y of X * Y mod P greater than */
00574             #undef ESP_RSA_MULM_BITS
00575             #define ESP_RSA_MULM_BITS 16
00576         #endif
00577     #endif
00578 #endif
00579
00580
00581 /* Pause in a loop rather than exit. */
00582 #define WOLFSSL_ESPIDF_ERROR_PAUSE
00583
00584 #define NO_HW_MATH_TEST
00585

```

```

00586 /* for test.c */
00587 /* #define HASH_SIZE_LIMIT */
00588
00589 /* Optionally turn off HW math checks */
00590 /* #define NO_HW_MATH_TEST */
00591
00592 /* Optionally include alternate HW test library: alt_hw_test.h */
00593 /* When enabling, the ./components/wolfssl/CMakeLists.txt file
00594  * will need the name of the library in the idf_component_register
00595  * for the PRIV_REQUIRES list. */
00596 /* #define INCLUDE_ALT_HW_TEST */
00597
00598 /* optionally turn off individual math HW acceleration features */
00599
00600 /* Turn off Large Number ESP32 HW Multiplication:
00601 ** [Z = X * Y] in esp_mp_mul() */
00602 /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MP_MUL */
00603
00604 /* Turn off Large Number ESP32 HW Modular Exponentiation:
00605 ** [Z = X^Y mod M] in esp_mp_exptmod() */
00606 /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_EXPTMOD */
00607
00608 /* Turn off Large Number ESP32 HW Modular Multiplication
00609 ** [Z = X * Y mod M] in esp_mp_mulmod() */
00610 /* #define NO_WOLFSSL_ESP32_CRYPT_RSA_PRI_MULMOD */
00611
00612
00613 /* used by benchmark: */
00614 #define WOLFSSL_PUBLIC_MP
00615
00616 /* when turning on ECC508 / ECC608 support
00617 #define WOLFSSL_ESPROOM32SE
00618 #define HAVE_PK_CALLBACKS
00619 #define WOLFSSL_ATECC508A
00620 #define ATCA_WOLFSSL
00621 */
00622
00623 /***** Certificate Macros *****/
00624 *
00625 * The section below defines macros used in typically all of the wolfSSL
00626 * examples such as the client and server for certs stored in header files.
00627 *
00628 * There are various certificate examples in this header file:
00629 * https://github.com/wolfSSL/wolfssl/blob/master/wolfssl/certs_test.h
00630 *
00631 * To use the sets of macros below, define *one* of these:
00632 *
00633 *     USE_CERT_BUFFERS_1024 - ECC 1024 bit encoded ASN1
00634 *     USE_CERT_BUFFERS_2048 - RSA 2048 bit encoded ASN1
00635 *     WOLFSSL_SM[2,3,4] - SM Ciphers
00636 *
00637 * For example: define USE_CERT_BUFFERS_2048 to use CA Certs used in this
00638 * wolfSSL function for the `ca_cert_der_2048` buffer, size and types:
00639 *
00640 *     ret = wolfSSL_CTX_load_verify_buffer(ctx,
00641 *     CTX_CA_CERT,
00642 *     CTX_CA_CERT_SIZE,
00643 *     CTX_CA_CERT_TYPE);
00644 *
00645 * See
https://www.wolfssl.com/documentation/manuals/wolfssl/group\_\_CertsKeys.html#function-wolfssl\_ctx\_load\_verify\_buffer
00646 *
00647 * In this case the CTX_CA_CERT will be defined as `ca_cert_der_2048` as
00648 * defined here: https://github.com/wolfSSL/wolfssl/blob/master/wolfssl/certs\_test.h
00649 *
00650 * The CTX_CA_CERT_SIZE and CTX_CA_CERT_TYPE are similarly used to reference
00651 * array size and cert type respectively.
00652 *
00653 * Similarly for loading the private client key:
00654 *
00655 *     ret = wolfSSL_CTX_use_PrivateKey_buffer(ctx,
00656 *     CTX_CLIENT_KEY,
00657 *     CTX_CLIENT_KEY_SIZE,
00658 *     CTX_CLIENT_KEY_TYPE);
00659 *
00660 * see
https://www.wolfssl.com/documentation/manuals/wolfssl/group\_\_CertsKeys.html#function-wolfssl\_ctx\_use\_privatekey\_buffer
00661 *
00662 * Similarly, the other macros are for server certificates and keys:
00663 *     `CTX_SERVER_CERT` and `CTX_SERVER_KEY` are available.
00664 *
00665 * The certificate and key names are typically `static const unsigned char`
00666 * arrays. The [NAME]_size are typically `sizeof([array name])`, and the types
00667 * are the known wolfSSL encoding type integers (e.g. WOLFSSL_FILETYPE_PEM).
00668 *
00669 * See `SSL_FILETYPE_[name]` in
00670 * https://github.com/wolfSSL/wolfssl/blob/master/wolfssl/ssl.h

```

```

00671 *
00672 * See Abstract Syntax Notation One (ASN.1) in:
00673 *   https://github.com/wolfSSL/wolfssl/blob/master/wolfssl/wolfcrypt/asn.h
00674 *
00675 * Optional SM4 Ciphers:
00676 *
00677 * Although the SM ciphers are shown here, the `certs_test_sm.h` may not yet
00678 * be available. See:
00679 *   https://github.com/wolfSSL/wolfssl/pull/6825
00680 *   https://github.com/wolfSSL/wolfsm
00681 *
00682 * Uncomment these 3 macros to enable the SM Ciphers and use the macros below.
00683 */
00684
00685 /*
00686 #define WOLFSSL_SM2
00687 #define WOLFSSL_SM3
00688 #define WOLFSSL_SM4
00689 */
00690
00691 /* Conditional macros used in wolfSSL TLS client and server examples */
00692 #if defined(WOLFSSL_SM2) || defined(WOLFSSL_SM3) || defined(WOLFSSL_SM4)
00693     #include <wolfssl/certs_test_sm.h>
00694     #define CTX_CA_CERT            root_sm2
00695     #define CTX_CA_CERT_SIZE      sizeof_root_sm2
00696     #define CTX_CA_CERT_TYPE      WOLFSSL_FILETYPE_PEM
00697     #define CTX_SERVER_CERT       server_sm2
00698     #define CTX_SERVER_CERT_SIZE  sizeof_server_sm2
00699     #define CTX_SERVER_CERT_TYPE  WOLFSSL_FILETYPE_PEM
00700     #define CTX_SERVER_KEY        server_sm2_priv
00701     #define CTX_SERVER_KEY_SIZE   sizeof_server_sm2_priv
00702     #define CTX_SERVER_KEY_TYPE   WOLFSSL_FILETYPE_PEM
00703
00704     #undef WOLFSSL_BASE16
00705     #define WOLFSSL_BASE16
00706 #else
00707     #if defined(USE_CERT_BUFFERS_2048)
00708         /* Be sure to include in app when using example certs: */
00709         /* #include <wolfssl/certs_test.h> */
00710         #define CTX_CA_CERT        ca_cert_der_2048
00711         #define CTX_CA_CERT_SIZE   sizeof_ca_cert_der_2048
00712         #define CTX_CA_CERT_TYPE   WOLFSSL_FILETYPE_ASN1
00713
00714         #define CTX_SERVER_CERT    server_cert_der_2048
00715         #define CTX_SERVER_CERT_SIZE sizeof_server_cert_der_2048
00716         #define CTX_SERVER_CERT_TYPE WOLFSSL_FILETYPE_ASN1
00717         #define CTX_SERVER_KEY     server_key_der_2048
00718         #define CTX_SERVER_KEY_SIZE sizeof_server_key_der_2048
00719         #define CTX_SERVER_KEY_TYPE WOLFSSL_FILETYPE_ASN1
00720
00721         #define CTX_CLIENT_CERT    client_cert_der_2048
00722         #define CTX_CLIENT_CERT_SIZE sizeof_client_cert_der_2048
00723         #define CTX_CLIENT_CERT_TYPE WOLFSSL_FILETYPE_ASN1
00724         #define CTX_CLIENT_KEY     client_key_der_2048
00725         #define CTX_CLIENT_KEY_SIZE sizeof_client_key_der_2048
00726         #define CTX_CLIENT_KEY_TYPE WOLFSSL_FILETYPE_ASN1
00727
00728     #elif defined(USE_CERT_BUFFERS_1024)
00729         /* Be sure to include in app when using example certs: */
00730         /* #include <wolfssl/certs_test.h> */
00731         #define CTX_CA_CERT        ca_cert_der_1024
00732         #define CTX_CA_CERT_SIZE   sizeof_ca_cert_der_1024
00733         #define CTX_CA_CERT_TYPE   WOLFSSL_FILETYPE_ASN1
00734
00735         #define CTX_CLIENT_CERT    client_cert_der_1024
00736         #define CTX_CLIENT_CERT_SIZE sizeof_client_cert_der_1024
00737         #define CTX_CLIENT_CERT_TYPE WOLFSSL_FILETYPE_ASN1
00738         #define CTX_CLIENT_KEY     client_key_der_1024
00739         #define CTX_CLIENT_KEY_SIZE sizeof_client_key_der_1024
00740         #define CTX_CLIENT_KEY_TYPE WOLFSSL_FILETYPE_ASN1
00741
00742         #define CTX_SERVER_CERT    server_cert_der_1024
00743         #define CTX_SERVER_CERT_SIZE sizeof_server_cert_der_1024
00744         #define CTX_SERVER_CERT_TYPE WOLFSSL_FILETYPE_ASN1
00745         #define CTX_SERVER_KEY     server_key_der_1024
00746         #define CTX_SERVER_KEY_SIZE sizeof_server_key_der_1024
00747         #define CTX_SERVER_KEY_TYPE WOLFSSL_FILETYPE_ASN1
00748     #else
00749         /* Optionally define custom cert arrays, sizes, and types here */
00750         #error "Must define USE_CERT_BUFFERS_2048 or USE_CERT_BUFFERS_1024"
00751     #endif
00752 #endif /* Conditional key and cert constant names */
00753
00754 /*****
00755 ** Sanity Checks
00756 *****/
00757 #if defined(CONFIG_ESP_MAIN_TASK_STACK_SIZE)

```

```
00758     #if defined(WOLFCRYPT_HAVE_SRP)
00759     #if defined(FP_MAX_BITS)
00760         #if FP_MAX_BITS < (8192 * 2)
00761             #define ESP_SRP_MINIMUM_STACK_8K (24 * 1024)
00762         #else
00763             #define ESP_SRP_MINIMUM_STACK_8K (28 * 1024)
00764         #endif
00765     #else
00766         #error "Please define FP_MAX_BITS when using WOLFCRYPT_HAVE_SRP."
00767     #endif
00768
00769     #if (CONFIG_ESP_MAIN_TASK_STACK_SIZE < ESP_SRP_MINIMUM_STACK)
00770         #warning "WOLFCRYPT_HAVE_SRP enabled with small stack size"
00771     #endif
00772 #endif
00773 #else
00774     #warning "CONFIG_ESP_MAIN_TASK_STACK_SIZE not defined!"
00775 #endif
```

3.24 D:/Dokumente/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:/Dokumente/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

<i>quorum</i>	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/loT-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

<i>quorum</i>	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.](#)

```
00001 #ifndef TEST_PERFORMANCE_H
00002 #define TEST_PERFORMANCE_H
00003 #include "test_performance.h"
00004 #include "model.h"
00005 #include "crypto_utils.h"
00006 #include "freertos/task.h"
00007 #include "freertos/FreeRTOS.h"
00008 #include "esp_task_wdt.h"
00009 #include "esp_timer.h"
00010 #include <math.h>
00011 #include "sdkconfig.h"
00012 #include "esp_task_wdt.h"
00013
00014 void perform_measurements_keygen(int quorum);
00015 void perform_measurements_backup();
00016 void perform_measurements_verification();
00017
00018 #endif // TEST_PERFORMANCE_H
```

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

Index

adapter.c
 backup_count, [4](#)
 backup_map, [4](#)
 guardian, [4](#)
 mac, [4](#)
 max_guardians, [4](#)
 mqtt_app_start, [4](#)
 mqtt_event_handler, [4](#)
 pubkey_count, [5](#)
 pubkey_map, [5](#)
 quorum, [5](#)
adapter.h
 mqtt_app_start, [6](#)
app_main
 main.c, [78](#)

backup_count
 adapter.c, [4](#)
backup_map
 adapter.c, [4](#)
BENCH_EMBEDDED
 user_settings.h, [58](#)
BLOCK_SIZE
 crypto_utils.h, [38](#)

ChaumPedersenProof, [37](#)
CiphertextDecryptionContest, [37](#)
CiphertextDecryptionSelection, [37](#)
CiphertextTally, [37](#)
CiphertextTallyContest, [37](#)
CiphertextTallySelection, [36](#)
Coefficient, [35](#)
combine_election_public_keys
 model.c, [21](#)
 model.h, [17](#)
compute_decryption_share
 model.c, [22](#)
 model.h, [18](#)
compute_decryption_share_for_contest
 crypto_utils.c, [26](#)
 crypto_utils.h, [38](#)
compute_polynomial_coordinate
 crypto_utils.c, [26](#)
 crypto_utils.h, [38](#)
constants.c
 g_3072, [25](#)
 p_3072, [25](#)
 q_256, [25](#)
constants.h
 g_3072, [33](#)
 p_3072, [33](#)
 q_256, [33](#)
crypto_utils.c
 compute_decryption_share_for_contest, [26](#)
 compute_polynomial_coordinate, [26](#)
 elgamal_combine_public_keys, [27](#)
 generate_polynomial, [27](#)
 hash, [28](#)
 hash_keys, [28](#)
 hashed_elgamal_decrypt, [29](#)
 hashed_elgamal_encrypt, [29](#)
 make_chaum_pedersen, [30](#)
 print_byte_array, [31](#)
 print_sp_int, [31](#)
 rand_q, [31](#)
 verify_chaum_pedersen, [32](#)
 verify_polynomial_coordinate, [32](#)
crypto_utils.h
 BLOCK_SIZE, [38](#)
 compute_decryption_share_for_contest, [38](#)
 compute_polynomial_coordinate, [38](#)
 elgamal_combine_public_keys, [39](#)
 generate_polynomial, [39](#)
 hash, [40](#)
 hash_keys, [41](#)
 hashed_elgamal_decrypt, [41](#)
 hashed_elgamal_encrypt, [42](#)
 make_chaum_pedersen, [42](#)
 MODIFIED_CHUNK_SIZE, [38](#)
 print_byte_array, [43](#)
 print_sp_int, [43](#)
 rand_q, [44](#)
 verify_polynomial_coordinate, [44](#)
CTX_CA_CERT
 user_settings.h, [58](#)
CTX_CA_CERT_SIZE
 user_settings.h, [58](#)
CTX_CA_CERT_TYPE
 user_settings.h, [58](#)
CTX_CLIENT_CERT
 user_settings.h, [58](#)
CTX_CLIENT_CERT_SIZE
 user_settings.h, [58](#)
CTX_CLIENT_CERT_TYPE
 user_settings.h, [58](#)
CTX_CLIENT_KEY
 user_settings.h, [58](#)
CTX_CLIENT_KEY_SIZE
 user_settings.h, [58](#)
CTX_CLIENT_KEY_TYPE
 user_settings.h, [59](#)
CTX_SERVER_CERT
 user_settings.h, [59](#)
CTX_SERVER_CERT_SIZE
 user_settings.h, [59](#)
CTX_SERVER_CERT_TYPE
 user_settings.h, [59](#)
CTX_SERVER_KEY
 user_settings.h, [59](#)
CTX_SERVER_KEY_SIZE
 user_settings.h, [59](#)

- generate_polynomial
 - crypto_utils.c, [27](#)
 - crypto_utils.h, [39](#)
- guardian
 - adapter.c, [4](#)
- Guardian - ElectionGuard ESP32 Client, [1](#)
- hash
 - crypto_utils.c, [28](#)
 - crypto_utils.h, [40](#)
- hash_keys
 - crypto_utils.c, [28](#)
 - crypto_utils.h, [41](#)
- HASH_SIZE_LIMIT
 - user_settings.h, [60](#)
- hashed_elgamal_decrypt
 - crypto_utils.c, [29](#)
 - crypto_utils.h, [41](#)
- hashed_elgamal_encrypt
 - crypto_utils.c, [29](#)
 - crypto_utils.h, [42](#)
- HashedElGamalCiphertext, [36](#)
- HAVE_AEAD
 - user_settings.h, [60](#)
- HAVE_AESGCM
 - user_settings.h, [60](#)
- HAVE_BLAKE2
 - user_settings.h, [60](#)
- HAVE_BLAKE2B
 - user_settings.h, [60](#)
- HAVE_BLAKE2S
 - user_settings.h, [60](#)
- HAVE_CURVE25519
 - user_settings.h, [61](#)
- HAVE_DH
 - user_settings.h, [61](#)
- HAVE_DSA
 - user_settings.h, [61](#)
- HAVE_ECC
 - user_settings.h, [61](#)
- HAVE_ED25519
 - user_settings.h, [61](#)
- HAVE_FFDHE
 - user_settings.h, [61](#)
- HAVE_FFDHE_2048
 - user_settings.h, [61](#)
- HAVE_HKDF
 - user_settings.h, [61](#)
- HAVE_HPKE
 - user_settings.h, [61](#)
- HAVE_POLY1305
 - user_settings.h, [61](#)
- HAVE_SESSION_TICKET
 - user_settings.h, [62](#)
- HAVE_SUPPORTED_CURVES
 - user_settings.h, [62](#)
- HAVE_TLS_EXTENSIONS
 - user_settings.h, [62](#)
- HAVE_VERSION_EXTENDED_INFO
 - user_settings.h, [62](#)
- HAVE_X963_KDF
 - user_settings.h, [62](#)
- mac
 - adapter.c, [4](#)
- main.c
 - app_main, [78](#)
- make_chaum_pedersen
 - crypto_utils.c, [30](#)
 - crypto_utils.h, [42](#)
- max_guardians
 - adapter.c, [4](#)
- MEASUREMENT_RUNS
 - test_decrypt.c, [16](#)
 - test_performance.c, [79](#)
- model.c
 - combine_election_public_keys, [21](#)
 - compute_decryption_share, [22](#)
 - generate_election_key_pair, [22](#)
 - generate_election_partial_key_backup, [23](#)
 - verify_election_partial_key_backup, [24](#)
- model.h
 - combine_election_public_keys, [17](#)
 - compute_decryption_share, [18](#)
 - generate_election_key_pair, [18](#)
 - generate_election_partial_key_backup, [19](#)
 - test_hash, [20](#)
 - verify_election_partial_key_backup, [20](#)
- MODIFIED_CHUNK_SIZE
 - crypto_utils.h, [38](#)
- mqtt_app_start
 - adapter.c, [4](#)
 - adapter.h, [6](#)
- mqtt_event_handler
 - adapter.c, [4](#)
- NO_CRYPT_BENCHMARK
 - user_settings.h, [62](#)
- NO_CRYPT_TEST
 - user_settings.h, [62](#)
- NO_DSA
 - user_settings.h, [62](#)
- NO_ESP32_CRYPT
 - user_settings.h, [62](#)
- NO_ESP_SDK_WIFI
 - user_settings.h, [62](#)
- NO_FILESYSTEM
 - user_settings.h, [63](#)
- NO_HW_MATH_TEST
 - user_settings.h, [63](#)
- NO_MD4
 - user_settings.h, [63](#)
- NO_MD5
 - user_settings.h, [63](#)
- NO_OLD_TLS
 - user_settings.h, [63](#)
- NO_PSK
 - user_settings.h, [63](#)

- NO_PWDBASED
 - user_settings.h, 63
- NO_RC4
 - user_settings.h, 63
- NO_SESSION_CACHE
 - user_settings.h, 63
- NO_SHA
 - user_settings.h, 64
- NO_TLS
 - user_settings.h, 64
- NO_WOLFSSL_ESP32_CRYPT_AES
 - user_settings.h, 64
- NO_WOLFSSL_ESP32_CRYPT_HASH
 - user_settings.h, 64
- NO_WOLFSSL_ESP32_CRYPT_RSA_PRI
 - user_settings.h, 64
- OPENSSL_EXTRA
 - user_settings.h, 64
- p_3072
 - constants.c, 25
 - constants.h, 33
- perform_measurement
 - test_decrypt.c, 16
 - test_decrypt.h, 11
- perform_measurements_backup
 - test_performance.c, 79
 - test_performance.h, 80
- perform_measurements_keygen
 - test_performance.c, 79
 - test_performance.h, 80
- perform_measurements_verification
 - test_performance.c, 79
 - test_performance.h, 80
- print_byte_array
 - crypto_utils.c, 31
 - crypto_utils.h, 43
- print_sp_int
 - crypto_utils.c, 31
 - crypto_utils.h, 43
- pubkey_count
 - adapter.c, 5
- pubkey_map
 - adapter.c, 5
- q_256
 - constants.c, 25
 - constants.h, 33
- quorum
 - adapter.c, 5
- rand_q
 - crypto_utils.c, 31
 - crypto_utils.h, 44
- RSA_LOW_MEM
 - user_settings.h, 64
- SchnorrProof, 35
- SCRIPT_TEST_ALL
 - user_settings.h, 64
- serialize.c
 - deserialize_ciphertext_tally, 12
 - deserialize_election_key_pair, 13
 - deserialize_election_partial_key_backup, 13
 - deserialize_election_partial_key_verification, 14
 - serialize_DecryptionShare, 14
 - serialize_election_key_pair, 14
 - serialize_election_partial_key_backup, 15
 - serialize_election_partial_key_verification, 15
- serialize.h
 - deserialize_ciphertext_tally, 7
 - deserialize_election_key_pair, 7
 - deserialize_election_partial_key_backup, 8
 - deserialize_election_partial_key_verification, 8
 - serialize_DecryptionShare, 9
 - serialize_election_key_pair, 9
 - serialize_election_partial_key_backup, 10
 - serialize_election_partial_key_verification, 10
- serialize_DecryptionShare
 - serialize.c, 14
 - serialize.h, 9
- serialize_election_key_pair
 - serialize.c, 14
 - serialize.h, 9
- serialize_election_partial_key_backup
 - serialize.c, 15
 - serialize.h, 10
- serialize_election_partial_key_verification
 - serialize.c, 15
 - serialize.h, 10
- SP_MATH
 - user_settings.h, 64
- test_decrypt.c
 - MEASUREMENT_RUNS, 16
 - perform_measurement, 16
- test_decrypt.h
 - perform_measurement, 11
- TEST_ESPIDF_ALL_WOLFSSL
 - user_settings.h, 64
- test_hash
 - model.h, 20
- test_performance.c
 - MEASUREMENT_RUNS, 79
 - perform_measurements_backup, 79
 - perform_measurements_keygen, 79
 - perform_measurements_verification, 79
- test_performance.h
 - perform_measurements_backup, 80
 - perform_measurements_keygen, 80
 - perform_measurements_verification, 80
- TFM_TIMING_RESISTANT
 - user_settings.h, 65
- USE_CERT_BUFFERS_2048
 - user_settings.h, 65
- USE_CERT_BUFFERS_256

- user_settings.h, 65
- USE_FAST_MATH
 - user_settings.h, 65
- user_settings.h
 - BENCH_EMBEDDED, 58
 - CTX_CA_CERT, 58
 - CTX_CA_CERT_SIZE, 58
 - CTX_CA_CERT_TYPE, 58
 - CTX_CLIENT_CERT, 58
 - CTX_CLIENT_CERT_SIZE, 58
 - CTX_CLIENT_CERT_TYPE, 58
 - CTX_CLIENT_KEY, 58
 - CTX_CLIENT_KEY_SIZE, 58
 - CTX_CLIENT_KEY_TYPE, 59
 - CTX_SERVER_CERT, 59
 - CTX_SERVER_CERT_SIZE, 59
 - CTX_SERVER_CERT_TYPE, 59
 - CTX_SERVER_KEY, 59
 - CTX_SERVER_KEY_SIZE, 59
 - CTX_SERVER_KEY_TYPE, 59
 - CURVE25519_SMALL, 59
 - DEBUG_WOLFSSL_MALLOCC, 59
 - ECC_SHAMIR, 59
 - ESP_RSA_TIMEOUT_CNT, 60
 - FP_MAX_BITS, 60
 - HASH_SIZE_LIMIT, 60
 - HAVE_AEAD, 60
 - HAVE_AESGCM, 60
 - HAVE_BLAKE2, 60
 - HAVE_BLAKE2B, 60
 - HAVE_BLAKE2S, 60
 - HAVE_CURVE25519, 61
 - HAVE_DH, 61
 - HAVE_DSA, 61
 - HAVE_ECC, 61
 - HAVE_ED25519, 61
 - HAVE_FFDHE, 61
 - HAVE_FFDHE_2048, 61
 - HAVE_HKDF, 61
 - HAVE_HPKE, 61
 - HAVE_POLY1305, 61
 - HAVE_SESSION_TICKET, 62
 - HAVE_SUPPORTED_CURVES, 62
 - HAVE_TLS_EXTENSIONS, 62
 - HAVE_VERSION_EXTENDED_INFO, 62
 - HAVE_X963_KDF, 62
 - NO_CRYPT_BENCHMARK, 62
 - NO_CRYPT_TEST, 62
 - NO_DSA, 62
 - NO_ESP32_CRYPT, 62
 - NO_ESP_SDK_WIFI, 62
 - NO_FILESYSTEM, 63
 - NO_HW_MATH_TEST, 63
 - NO_MD4, 63
 - NO_MD5, 63
 - NO_OLD_TLS, 63
 - NO_PSK, 63
 - NO_PWDBASED, 63
 - NO_RC4, 63
 - NO_SESSION_CACHE, 63
 - NO_SHA, 64
 - NO_TLS, 64
 - NO_WOLFSSL_ESP32_CRYPT_AES, 64
 - NO_WOLFSSL_ESP32_CRYPT_HASH, 64
 - NO_WOLFSSL_ESP32_CRYPT_RSA_PRI, 64
 - OPENSSL_EXTRA, 64
 - RSA_LOW_MEM, 64
 - SCRYPT_TEST_ALL, 64
 - SP_MATH, 64
 - TEST_ESPIDF_ALL_WOLFSSL, 64
 - TFM_TIMING_RESISTANT, 65
 - USE_CERT_BUFFERS_2048, 65
 - USE_CERT_BUFFERS_256, 65
 - USE_FAST_MATH, 65
 - WC_KDF_NIST_SP_800_56C, 65
 - WC_RC2, 65
 - WC_RSA_PSS, 65
 - WOLFCRYPT_HAVE_SRP, 65
 - WOLFSSL_AES_128, 65
 - WOLFSSL_AES_CFB, 65
 - WOLFSSL_AES_COUNTER, 66
 - WOLFSSL_AES_EAX, 66
 - WOLFSSL_AES_OFB, 66
 - WOLFSSL_AES_SIV, 66
 - WOLFSSL_AES_XTS, 66
 - WOLFSSL_ALLOW_RC4, 66
 - WOLFSSL_ASN_TEMPLATE, 66
 - WOLFSSL_BENCHMARK_FIXED_UNITS_KB, 66
 - WOLFSSL_CERT_EXT, 66
 - WOLFSSL_CERT_GEN, 66
 - WOLFSSL_CERT_PIV, 67
 - WOLFSSL_CERT_REQ, 67
 - WOLFSSL_CERT_TEXT, 67
 - WOLFSSL_CMAC, 67
 - WOLFSSL_ESPIDF, 67
 - WOLFSSL_ESPIDF_ERROR_PAUSE, 67
 - WOLFSSL_HAVE_SP_RSA, 67
 - WOLFSSL_KEY_GEN, 67
 - WOLFSSL_MD2, 67
 - WOLFSSL_PUBLIC_MP, 67
 - WOLFSSL_SHA512, 68
 - WOLFSSL_SMALL_STACK, 68
 - WOLFSSL_SP_4096, 68
 - WOLFSSL_SP_LOW_MEM, 68
 - WOLFSSL_SP_MATH_ALL, 68
 - WOLFSSL_SP_NO_256, 68
 - WOLFSSL_SP_SMALL, 68
 - WOLFSSL_SYS_CA_CERTS, 68
 - WOLFSSL_TLS13, 68
 - WOLFSSL_WOLFSSH, 69
- utils.c
 - free_CharmPedersenProof, 52
 - free_CiphertextDecryptionContest, 52
 - free_CiphertextDecryptionSelection, 52
 - free_CiphertextTally, 53
 - free_CiphertextTallyContest, 53

- free_CiphertextTallySelection, 53
- free_Coefficient, 54
- free_DecryptionShare, 54
- free_ElectionKeyPair, 54
- free_ElectionPartialKeyPairBackup, 55
- free_ElectionPolynomial, 55
- free_SchnorrProof, 55
- utils.h
 - free_ChaumPedersenProof, 47
 - free_CiphertextDecryptionContest, 47
 - free_CiphertextDecryptionSelection, 48
 - free_CiphertextTally, 48
 - free_CiphertextTallyContest, 48
 - free_CiphertextTallySelection, 49
 - free_Coefficient, 49
 - free_DecryptionShare, 49
 - free_ElectionKeyPair, 50
 - free_ElectionPartialKeyPairBackup, 50
 - free_ElectionPolynomial, 50
 - free_SchnorrProof, 50
- verify_chaum_pedersen
 - crypto_utils.c, 32
- verify_election_partial_key_backup
 - model.c, 24
 - model.h, 20
- verify_polynomial_coordinate
 - crypto_utils.c, 32
 - crypto_utils.h, 44
- WC_KDF_NIST_SP_800_56C
 - user_settings.h, 65
- WC_RC2
 - user_settings.h, 65
- WC_RSA_PSS
 - user_settings.h, 65
- WOLFCRYPT_HAVE_SRP
 - user_settings.h, 65
- WOLFSSL_AES_128
 - user_settings.h, 65
- WOLFSSL_AES_CFB
 - user_settings.h, 65
- WOLFSSL_AES_COUNTER
 - user_settings.h, 66
- WOLFSSL_AES_EAX
 - user_settings.h, 66
- WOLFSSL_AES_OFB
 - user_settings.h, 66
- WOLFSSL_AES_SIV
 - user_settings.h, 66
- WOLFSSL_AES_XTS
 - user_settings.h, 66
- WOLFSSL_ALLOW_RC4
 - user_settings.h, 66
- WOLFSSL_ASN_TEMPLATE
 - user_settings.h, 66
- WOLFSSL_BENCHMARK_FIXED_UNITS_KB
 - user_settings.h, 66
- WOLFSSL_CERT_EXT
 - user_settings.h, 66
- WOLFSSL_CERT_GEN
 - user_settings.h, 66
- WOLFSSL_CERT_PIV
 - user_settings.h, 67
- WOLFSSL_CERT_REQ
 - user_settings.h, 67
- WOLFSSL_CERT_TEXT
 - user_settings.h, 67
- WOLFSSL_CMAC
 - user_settings.h, 67
- WOLFSSL_ESPIDF
 - user_settings.h, 67
- WOLFSSL_ESPIDF_ERROR_PAUSE
 - user_settings.h, 67
- WOLFSSL_HAVE_SP_RSA
 - user_settings.h, 67
- WOLFSSL_KEY_GEN
 - user_settings.h, 67
- WOLFSSL_MD2
 - user_settings.h, 67
- WOLFSSL_PUBLIC_MP
 - user_settings.h, 67
- WOLFSSL_SHA512
 - user_settings.h, 68
- WOLFSSL_SMALL_STACK
 - user_settings.h, 68
- WOLFSSL_SP_4096
 - user_settings.h, 68
- WOLFSSL_SP_LOW_MEM
 - user_settings.h, 68
- WOLFSSL_SP_MATH_ALL
 - user_settings.h, 68
- WOLFSSL_SP_NO_256
 - user_settings.h, 68
- WOLFSSL_SP_SMALL
 - user_settings.h, 68
- WOLFSSL_SYS_CA_CERTS
 - user_settings.h, 68
- WOLFSSL_TLS13
 - user_settings.h, 68
- WOLFSSL_WOLFSSH
 - user_settings.h, 69