Cryptnox SDK for ESP32 Manual

Release 1.0.0

June 20, 2026

Chapter 1

cryptnox-sdk-esp32

1.0.0.1 cryptnox-sdk-esp32

ESP32 SDK for managing Cryptnox smart card wallets

`cryptnox-sdk-esp32` is an ESP-IDF component bundle that enables the use of **Cryptnox Smart cards** on ESP32-family platforms. It provides secure communication with the card, retrieves card information, and exposes basic cryptographic operations through the shared C++ core SDK.

1.0.1 Supported hardware

1.0.1.1 Cryptnox Hardware Wallet smart cards

Works with Cryptnox Hardware Wallet smart cards running firmware v1.6.0 or later.

Smart card	Wallet version
`Crypto Hardware Wallet - Dual Card Set`	v1.↩ 6.1

1.0.1.2 NFC readers

Reader	Type	Interface
`PN532 NFC Module`	Contactless (NFC/ISO 14443)	SPI or I^2C

1.0.1.3 Host board

Board	MCU	Notes
`ESP32-S3-DevKitC-1`	ESP32-S3-WROOM-1	Official Espressif dev kit — wire PN532 with jumpers to the 2×22-pin headers

1.0.2 Installation

1. Install **ESP-IDF v5.5** following the official guide.
2. Clone this repository **with submodules** (the C++ core SDK is a submodule):
   ```
   git clone --recurse-submodules https://github.com/cryptnox/cryptnox-sdk-esp32.git
   ```
3. From the project root, set the target and build:
   ```
   idf.py set-target esp32s3
   idf.py build flash monitor
   ```

1.0.3 Hardware setup

Attention

Always double-check the wiring before powering the board to prevent damage.

Wiring shown for the **ESP32-S3-DevKitC-1**.

1.0.3.1 ESP32-S3 and PN532 NFC — SPI interface

PN532 Pin	ESP32-S3 GPIO	Wire Color
VCC	3.3V	Red
GND	GND	Black
SCK	IO12	Blue
MISO	IO13	Green
MOSI	IO11	Yellow
SS (CS)	IO10	Violet

Important

Make sure the switches on the PN532 module are configured for **SPI** mode:

- **Switch 0** → HIGH
- **Switch 1** → LOW

1.0.3.2 ESP32-S3 and PN532 NFC — I^2C interface

PN532 Pin	ESP32-S3 GPIO	Wire Color
VCC	3.3V	Red
GND	GND	Black
SDA	IO8	Yellow
SCL	IO9	Blue
IRQ	IO20	Violet
RST	IO21	Grey

Important

Make sure the switches on the PN532 module are configured for **I²C** mode:

- **Switch 0** → LOW
- **Switch 1** → HIGH

1.0.4 Quick usage examples

All examples assume the shared SPI bus has been initialised and the three adapters (`ESP32Logger`, `ESP32CryptoProvider`, `ESP32Pn532Transport`) are wired into a `CryptnoxWallet`. Helper macros are used for brevity; see `examples/UsdcSigning/` for the full ESP-IDF boilerplate.

1.0.4.1 1. Connect to a Cryptnox card

```
#include "CryptnoxWallet.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "esp32_pn532_transport.h"
#include "esp_log.h"

static const char *TAG = "main";

extern "C" void app_main(void) {
    ESP32Logger          logger;
    ESP32CryptoProvider  crypto;
    ESP32Pn532Transport  transport(SPI2_HOST, /*cs_gpio=*/10);
    CryptnoxWallet       wallet(transport, logger, crypto);

    if (!wallet.begin()) {
        ESP_LOGE(TAG, "Reader not found");
        return;
    }

    CW_SecureSession session;
    if (!wallet.connect(session)) {
        ESP_LOGE(TAG, "Card not detected or secure channel failed");
        return;
    }

    CW_CardInfo info = {};
    if (wallet.getCardInfo(session, info)) {
        ESP_LOGI(TAG, "Card serial number: %s", info.serialNumber);
    }

    wallet.disconnect(session);
}
```

1.0.4.2 2. Verify the PIN code

The card must be initialised before calling `verifyPin`.

> **Security note:** Never hard-code a PIN in firmware — read it from a secure source at runtime and wipe the buffer with `CW_Utils::secure_wipe()` immediately after use.

```
extern "C" void app_main(void) {
    ESP32Logger          logger;
    ESP32CryptoProvider  crypto;
    ESP32Pn532Transport  transport(SPI2_HOST, /*cs_gpio=*/10);
    CryptnoxWallet       wallet(transport, logger, crypto);
    wallet.begin();

    CW_SecureSession session;
    if (!wallet.connect(session)) {
        ESP_LOGE(TAG, "Card not detected");
        return;
    }

    /* Read the PIN from a secure source -- never hard-code it. */
    uint8_t pin[CW_MAX_PIN_LENGTH] = { 0U };
    uint8_t pinLength = 0U;
    read_pin_from_secure_input(pin, &pinLength);  /* implement for your platform */

    CW_PinResult res = wallet.verifyPin(session, pin, pinLength);
```

```
    CW_Utils::secure_wipe(pin, sizeof(pin));  /* wipe PIN immediately after use */
    switch (res) {
        case CW_PIN_OK:
            ESP_LOGI(TAG, "PIN verified -- card is ready for signing");
            break;
        case CW_PIN_INVALID:
            ESP_LOGW(TAG, "Invalid PIN code");
            break;
        case CW_PIN_LENGTH_INVALID:
            ESP_LOGW(TAG, "Invalid PIN length or PIN authentication disabled");
            break;
        case CW_PIN_LOCKED:
            ESP_LOGW(TAG, "Card is locked -- power-cycle the card");
            break;
    }

    wallet.disconnect(session);
}
```

1.0.4.3 3. Sign a transaction hash

Signs a 32-byte digest (e.g. SHA-256 of a Bitcoin or Ethereum transaction) on the secp256k1 curve.

```
extern "C" void app_main(void) {
    ESP32Logger          logger;
    ESP32CryptoProvider  crypto;
    ESP32Pn532Transport  transport(SPI2_HOST, /*cs_gpio=*/10);
    CryptnoxWallet       wallet(transport, logger, crypto);
    wallet.begin();

    CW_SecureSession session;
    if (!wallet.connect(session)) {
        ESP_LOGE(TAG, "Card not detected");
        return;
    }

    const uint8_t hash[32] = { /* SHA-256(your_tx) */ };
    CW_SignRequest req(session,
                       /*keyType=*/    CW_KEY_SECP256K1,
                       /*signatureType=*/CW_SIG_ECDSA,
                       /*pinLessMode=*/false);
    req.hash         = hash;
    req.hashLength   = sizeof(hash);
    /* Read the PIN from a secure source -- never hard-code it in the firmware. */
    read_pin_from_secure_input(req.pin, NULL);  /* implement for your platform */

    CW_SignResult sig = wallet.sign(req);
    if (sig.errorCode == CW_SIGN_OK) {
        ESP_LOGI(TAG, "Signature OK (64-byte r||s in sig.signature)");
        // Forward sig.signature to your blockchain client...
    } else {
        ESP_LOGE(TAG, "Sign failed: 0x%02X", sig.errorCode);
    }

    wallet.disconnect(session);
}
```

See `examples/UsdcSigning/` for a full reference that wires up the SPI bus and polls the PN532.

1.0.5 Security

- **Flash Encryption and Secure Boot V2** must be enabled for production deployments. Defaults are set in `sdkconfig.defaults`. eFuse programming is irreversible — practice on a development board first.
- **Hard-coded secrets:** Never hard-code a PIN or private key in firmware. Wipe PIN buffers with `CW_Utils::secure_wipe()` immediately after use.
- **Dependency pinning:** Use the exact ESP-IDF tag (`v5.5.0`) and pin the `cryptnox-sdk-cpp` submodule to a commit hash.

1.0.6 Documentation

The generated documentation for this project is available here.

1.0.7 License

`cryptnox-sdk-esp32` is dual-licensed:

- **LGPL-3.0** for open-source projects and proprietary projects that comply with LGPL requirements
- **Commercial license** for projects that require a proprietary license without LGPL obligations (see COMMERCIAL.md for details)

For commercial inquiries, contact: `contact@cryptnox.com`

Chapter 2

Topic Documentation

2.1 PN532 NFC driver

Low-level PN532 driver (SPI + I^2C, ESP-IDF).

Classes

- struct pn532_config_t

  Compile-time configuration passed to pn532_init.
- struct pn532_t

  Opaque-like runtime state for a single PN532 instance.

Macros

- #define PN532_MIFARE_ISO14443A (0x00U)

  Baud-rate selector for ISO 14443-A (Mifare) cards passed to pn532_read_passive_target_id.
- #define PN532_MAX_APDU_LEN (252U)

  Maximum APDU payload length accepted by pn532_send_apdu.
- #define PN532_I2C_ADDRESS (0x24U)

  7-bit I^2C slave address of the PN532 (fixed in hardware).

Enumerations

- enum pn532_transport_t { PN532_TRANSPORT_SPI , PN532_TRANSPORT_I2C }

  Physical bus used to communicate with the PN532.

Functions

- esp_err_t pn532_init (pn532_t ∗dev, const pn532_config_t ∗config)

  Initialise the PN532 and bring it to a ready state.
- uint32_t pn532_get_firmware_version (pn532_t ∗dev)

  Query the PN532 firmware version.
- bool pn532_sam_config (pn532_t ∗dev)

  Configure the PN532's Security Access Module (SAM).
- uint32_t pn532_read_passive_target_id (pn532_t ∗dev, uint8_t cardbaudrate)

  Scan for a passive ISO 14443-A card and return its UID.
- bool pn532_send_apdu (pn532_t ∗dev, const uint8_t ∗apdu, uint8_t apdu_len, uint8_t ∗response, uint16_t ∗response_len)

  Exchange a single ISO-DEP APDU with the currently selected card.
- bool pn532_release_target (pn532_t ∗dev)

  Release the currently selected NFC target.

2.1.1 Detailed Description

Low-level PN532 driver (SPI + I²C, ESP-IDF).

2.1.2 Macro Definition Documentation

2.1.2.1 PN532_I2C_ADDRESS

```
#define PN532_I2C_ADDRESS (0x24U)
```

7-bit I²C slave address of the PN532 (fixed in hardware).
Definition at line 63 of file pn532.h.
Referenced by pn532_init_i2c().

2.1.2.2 PN532_MAX_APDU_LEN

```
#define PN532_MAX_APDU_LEN (252U)
```

Maximum APDU payload length accepted by pn532_send_apdu.
Constrained so that the PN532 InDataExchange frame LEN field (8-bit) never overflows: `cmd_total_len` = apdu_len + 2 ≤ 254, `frame_len` = 255.
Definition at line 60 of file pn532.h.
Referenced by pn532_send_apdu().

2.1.2.3 PN532_MIFARE_ISO14443A

```
#define PN532_MIFARE_ISO14443A (0x00U)
```

Baud-rate selector for ISO 14443-A (Mifare) cards passed to pn532_read_passive_target_id.
Definition at line 54 of file pn532.h.
Referenced by PN532Adapter::inListPassiveTarget(), and Pn532NfcTransport::inListPassiveTarget().

2.1.3 Enumeration Type Documentation

2.1.3.1 pn532_transport_t

```
enum pn532_transport_t
```

Physical bus used to communicate with the PN532.

Value	Bus	Typical use
`PN532_TRANSPORT_SPI`	SPI	ESP32-S3 dev kit + Keyestudio breakout
`PN532_TRANSPORT_I2C`	I²C	Cheap Yellow Display (CN1 connector)

Enumerator

PN532_TRANSPORT_SPI	SPI master via ESP-IDF `spi_master` driver.
PN532_TRANSPORT_I2C	I²C master via ESP-IDF v5.x `i2c_master` driver.

Definition at line 76 of file pn532.h.

2.1.4 Function Documentation

2.1.4.1 pn532_get_firmware_version()

```
uint32_t pn532_get_firmware_version (
            pn532_t * dev)
```

Query the PN532 firmware version.

Sends the `GetFirmwareVersion` command (0x02) and reads the response. Also serves as a FIFO-drain step after init to prevent stale bytes from confusing subsequent command sequences.

Parameters

`in`	*dev*	Initialised device handle.

Returns

Packed 32-bit version word: (IC << 24) | (Ver << 16) | (Rev << 8) | Support, or `0` on communication failure.

See also

pn532_init

Definition at line 718 of file pn532.c.
References pn532_buffer_equal(), PN532_BYTE_SHIFT_BITS, PN532_CMD_TIMEOUT_MS, PN532_FIRMWARE_CMD_LEN, PN532_FIRMWARE_HDR_LEN, PN532_FIRMWARE_RESP_LEN, PN532_FIRMWAREVERSION, PN532_FW_IC_OFFSET, PN532_FW_REV_OFFSET, PN532_FW_SUPPORT_OFFSE PN532_FW_VER_OFFSET, PN532_LOG_TAG, pn532_response_fw, read_data(), and send_command_check_ack().
Referenced by pn532_init(), PN532Adapter::printFirmwareVersion(), and Pn532NfcTransport::printFirmwareVersion().

2.1.4.2 pn532_init()

```
esp_err_t pn532_init (
            pn532_t * dev,
            const pn532_config_t * config)
```

Initialise the PN532 and bring it to a ready state.
Performs the full transport-specific bring-up:

- **SPI**: configures the SPI device, runs the wake-up byte sequence, sends a sacrificial pn532_sam_config (absorbs the post-power-on latency), then confirms readiness with pn532_get_firmware_version.
- **I²C**: configures the I²C bus and device, sends two SAMConfig frames to recover from Soft-Power-Down if the PN532 was already powered, then confirms readiness with pn532_get_firmware_version.

Parameters

`out`	*dev*	Device state structure to populate. Must remain valid for the lifetime of all subsequent calls on this device.
`in`	*config*	Transport and pin configuration. Caller may free or reuse this after `pn532_init` returns.

Returns

`ESP_OK` on success, an `esp_err_t` error code otherwise.

Note

The return value of the internal firmware-version and sacrificial SAMConfig calls is intentionally ignored; they are used purely for timing and FIFO-drain purposes.

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 657 of file pn532.c.
References pn532_get_firmware_version(), pn532_init_i2c(), pn532_init_spi(), PN532_LOG_TAG, pn532_sam_config(), PN532_SYNC_DELAY_MS, PN532_TRANSPORT_I2C, pn532_config_t::transport, and pn532_t::transport.
Referenced by app_main(), and PN532Adapter::begin().

2.1.4.3 pn532_read_passive_target_id()

```
uint32_t pn532_read_passive_target_id (
            pn532_t * dev,
            uint8_t cardbaudrate)
```

Scan for a passive ISO 14443-A card and return its UID.
Sends `InListPassiveTarget` (0x4A) with `maxTargets` = 1. Blocks until a card is found or the command times out inside the PN532 firmware.

Parameters

`in`	*dev*	Initialised device handle.
`in`	*cardbaudrate*	Baud-rate selector; use PN532_MIFARE_ISO14443A for standard Mifare / ISO 14443-A cards.

Returns

32-bit packed card UID (UID bytes packed MSB-first), or `0` if no card was found or a communication error occurred.

Note

UIDs longer than 4 bytes are truncated to the most-significant 32 bits.

Definition at line 775 of file pn532.c.
References PN532_BYTE_SHIFT_BITS, PN532_CMD_TIMEOUT_MS, PN532_INLISTPASSIVETARGET, PN532_PASSIVE_CMD_LEN, PN532_PASSIVE_EXPECTED_TARGETS, PN532_PASSIVE_MAX_TARGETS, PN532_PASSIVE_NUM_TARGETS_OFFSET, PN532_PASSIVE_RESP_LEN, PN532_PASSIVE_UID_DATA_OFFSET, PN532_PASSIVE_UID_LEN_OFFSET, read_data(), and send_command_check_ack().
Referenced by PN532Adapter::inListPassiveTarget(), and Pn532NfcTransport::inListPassiveTarget().

2.1.4.4 pn532_release_target()

```
bool pn532_release_target (
            pn532_t * dev)
```

Release the currently selected NFC target.
Sends `InRelease` (0x52) so the PN532 drops the logical target and is ready to list a new one. Should be called after finishing a card session before the next pn532_read_passive_target_id.

Parameters

`in`	*dev*	Initialised device handle.

Returns

`true` if the PN532 acknowledged the release successfully, `false` on communication failure.

Definition at line 875 of file pn532.c.
References PN532_CMD_TIMEOUT_MS, PN532_EXCHANGE_STATUS_OFFSET, PN532_EXCHANGE_STATUS_OK, PN532_EXCHANGE_TG, PN532_INRELEASE, PN532_INRELEASE_CMD_LEN, PN532_INRELEASE_RESP_LEN, read_data(), and send_command_check_ack().
Referenced by PN532Adapter::resetReader(), and Pn532NfcTransport::resetReader().

2.1.4.5 pn532_sam_config()

```
bool pn532_sam_config (
            pn532_t * dev)
```

Configure the PN532's Security Access Module (SAM).
Sends the `SAMConfiguration` command (0x14) with Normal Mode, a 1-second timeout, and IRQ enabled. Must be called (at least once) before listing passive targets or exchanging APDUs.
pn532_init already issues this command internally; application code calls it again via `CW_Nfc↩Transport::begin()` / Pn532NfcTransport::begin.

Parameters

`in`	*dev*	Initialised device handle.

Returns

`true` on success (PN532 acknowledged and returned response code `0x15`), `false` on timeout or unexpected response.

Definition at line 752 of file pn532.c.
References PN532_CMD_TIMEOUT_MS, PN532_SAM_CMD_LEN, PN532_SAM_NORMAL_MODE, PN532_SAM_RESP_CODE, PN532_SAM_RESP_CODE_OFFSET, PN532_SAM_RESP_LEN, PN532_SAM_TIMEOUT, PN532_SAM_USE_IRQ, PN532_SAMCONFIGURATION, read_data(), and send_command_check_ack().
Referenced by Pn532NfcTransport::begin(), and pn532_init().

2.1.4.6 pn532_send_apdu()

```
bool pn532_send_apdu (
            pn532_t * dev,
            const uint8_t * apdu,
            uint8_t apdu_len,
            uint8_t * response,
            uint16_t * response_len)
```

Exchange a single ISO-DEP APDU with the currently selected card.
Wraps the PN532 `InDataExchange` command (0x40). Handles both normal PN532 frames (LEN $\leq$ 255) and extended frames (LEN $>$ 255) transparently; the frame format is detected from the response header.

Parameters

`in`	*dev*	Initialised device handle.
`in`	*apdu*	APDU command bytes (must not be `NULL`).
`in`	*apdu_len*	Length of `apdu`; must be $\leq$ PN532_MAX_APDU_LEN.
`out`	*response*	Caller-allocated buffer for the card's DataOut.
`in,out`	*response_len*	In: capacity of `response` in bytes. Out: number of DataOut bytes actually written.

Returns

`true` on success (PN532 error byte = 0x00), `false` on a PN532 transport error, a card-level error, or if `apdu_len` exceeds PN532_MAX_APDU_LEN.

Warning

`response` must be large enough to hold the full DataOut; if capacity is exceeded the copy is silently truncated to `*response_len`.

See also

pn532_release_target

Definition at line 805 of file pn532.c.
References PN532_APDU_TIMEOUT_MS, PN532_EXCHANGE_CMD_OVERHEAD, PN532_EXCHANGE_DATA_OFF
PN532_EXCHANGE_FRAME_MAX, PN532_EXCHANGE_LEN_BIAS, PN532_EXCHANGE_LEN_OFFSET,
PN532_EXCHANGE_STATUS_OFFSET, PN532_EXCHANGE_STATUS_OK, PN532_EXCHANGE_TG,
PN532_EXT_EXCHANGE_DATA_OFFSET, PN532_EXT_EXCHANGE_ERR_OFFSET,
PN532_EXT_FRAME_INDICATOR, PN532_EXT_FRAME_LENHI_OFFSET, PN532_EXT_FRAME_LENLO_OFFSET,
PN532_INDATAEXCHANGE, PN532_LOG_TAG, PN532_MAX_APDU_LEN, read_data_apdu_frame(),
and send_command_check_ack().
Referenced by PN532Adapter::sendAPDU(), Pn532NfcTransport::sendAPDU(), PN532Adapter::sendAPDULarge(),
and Pn532NfcTransport::sendAPDULarge().

2.2 ESP32 concrete adapters

Concrete CW_NfcTransport / CW_CryptoProvider / CW_Logger / CW_Platform implementations for ESP32 / ESP-IDF.

Files

- file esp32_crypto_provider.h
  CW_CryptoProvider implementation for ESP32 using mbedTLS and the hardware TRNG.
- file ESP32Logger.h
  CW_Logger implementation that writes to ESP32 UART0 via `printf`.
- file ESP32Platform.h
  CW_Platform implementation for ESP32 using FreeRTOS.
- file pn532_adapter.h
  Self-contained CW_NfcTransport adapter that owns its pn532_t handle.
- file Pn532NfcTransport.h
  CW_NfcTransport adapter wrapping the ESP-IDF PN532 NFC driver.

Classes

- class ESP32CryptoProvider
  CW_CryptoProvider backed by mbedTLS and the ESP32 hardware TRNG.
- class ESP32Logger
  CW_Logger backed by ESP32 UART0.
- class ESP32Platform
  CW_Platform backed by FreeRTOS `vTaskDelay`.
- class PN532Adapter
  Self-contained CW_NfcTransport that owns its pn532_t instance.

2.2.1 Detailed Description

Concrete CW_NfcTransport / CW_CryptoProvider / CW_Logger / CW_Platform implementations for ESP32 / ESP-IDF.
Each class in this group implements one of the abstract interfaces declared in adapters using the ESP-IDF peripheral drivers and mbedTLS:

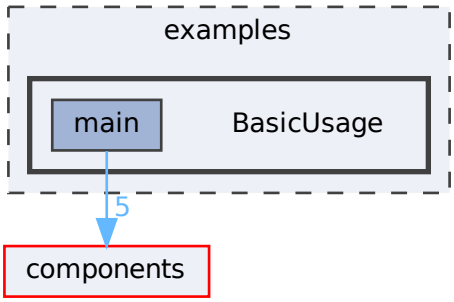
Adapter	Interface	Backing library / hardware
Pn532NfcTransport	CW_NfcTransport	ESP-IDF PN532 C driver (injected handle)
PN532Adapter	CW_NfcTransport	ESP-IDF PN532 C driver (owned handle)
ESP32CryptoProvider	CW_CryptoProvider	mbedTLS + ESP32 hardware TRNG
ESP32Logger	CW_Logger	ESP32 UART0 (development builds)
ESP32Platform	CW_Platform	FreeRTOS `vTaskDelay`

Chapter 3

Directory Documentation

3.1 examples/BasicUsage Directory Reference

Directory dependency graph for BasicUsage:



Directories

- directory main

Files

- file config.template.h

3.1.1 Detailed Description

3.1.1.0.1 cryptnox-sdk-esp32

ESP32 SDK for managing Cryptnox smart card wallets

3.1.2 BasicUsage — End-to-End Walkthrough (SPI or I²C)

A single self-contained ESP-IDF project that exercises the **full Cryptnox flow** on ESP32: pick SPI or I²C at build time, open the secure channel, sign a 32-byte hash, wipe the secrets, disconnect. Reads as a checklist of every step a production firmware will perform.
If you only need **one** of the steps, see the focused examples:

You want…	See
The secure channel + card identity	Connect
A PIN verification flow	VerifyPin
A signature without the rest	Sign
A real Ethereum tx broadcast	UsdcSigning

3.1.2.1 Requirements

Component	Details
Hardware Wallet	Cryptnox Hardware Wallet, initialised **and** seeded
NFC reader	PN532 wired on **SPI** (default) or **I²C** — see `hardware setup`
Board	ESP32-S3-DevKitC-1
Toolchain	ESP-IDF v5.5

3.1.2.2 Quick start

1. **Pick the bus** at the top of `main/main.cpp` — set exactly one of:

```
#define SPI_ENABLED  1     // MOSI=11, MISO=13, SCLK=12, CS=10
#define I2C_ENABLED  0     // (set to 1 for I²C; mutually exclusive)
```

2. **Set the PIN** to match `cryptnox init`:

```
static const uint8_t DEFAULT_PIN[]   = "000000000";
static const size_t  DEFAULT_PIN_LEN = sizeof(DEFAULT_PIN) - 1U;
```

3. **Create your runtime config from the template** (kept out of git so credentials never get committed) and fill in Wi-Fi:

```
cp config.template.h main/config.h
# then edit main/config.h: set WIFI_SSID / WIFI_PASSWORD
```

4. Build, flash and monitor:

```
cd examples/BasicUsage
idf.py set-target esp32s3                # once
idf.py build flash monitor
```

5. Exit the monitor with `Ctrl-]`. Place the card on the PN532 antenna.

3.1.2.2.1 Expected output

```
I (1280) basic_usage: Card connected and secure channel established
I (1290) basic_usage: Signing test hash...
I (1450) basic_usage: Signature received (64 bytes raw r||s)
I (1450) basic_usage: 7c 1f 3a 92 5e 0b 8c d4
I (1450) basic_usage: s:
I (1450) basic_usage: 12 e0 bc 4f a7 88 09 67
I (1450) basic_usage: Card processed successfully
```

Note

The SDK uses the PN532 extended-frame transport internally so manufacturer-certificate pages up to ∼411 bytes deliver correctly on both SPI and I²C.

3.1.2.3 How it works

```
app_main():
  nvs_flash_init()
  wifi_init() + esp_wifi_connect()        Wi-Fi → seeds the HW TRNG
  spi_bus_init()  /  i2c_bus_init()       Depending on SPI_ENABLED / I2C_ENABLED
  wallet.begin()                          PN532 reset + firmware probe

loop():
```

```
  wallet.connect(session)                SELECT + cert verify + ECDH
                                         + mutual authentication
  Build CW_SignRequest:
    keyType        = CW_SIGN_CURR_K1
    signatureType  = CW_SIGN_SIG_ECDSA_LOW_S
    pinLessMode    = CW_SIGN_WITH_PIN
    hash[32]       = 0x01 × 32             (test pattern)
    pin[]          = DEFAULT_PIN
  wallet.sign(req)                       SIGN APDU under the channel
  secure_wipe(hash, signature)           Zero local copies
  wallet.disconnect(session)             Zero session keys
  vTaskDelay(pdMS_TO_TICKS(1000))
```

3.1.2.4 Step-by-step code

1. Interface selection — exactly one of:

```
#define SPI_ENABLED  1
#define I2C_ENABLED  0
```

On `SPI_ENABLED == 1` the firmware brings up the SPI3 host with MOSI=11, MISO=13, SCLK=12, CS=10. On `I2C_ENABLED == 1` it brings up `I2C_NUM_0` with the pin map documented at the top of `main.cpp`.

2. Bring up the radio, bus, and reader:

```
ESP_ERROR_CHECK(nvs_flash_init());
wifi_init();                                  // brings Wi-Fi up → seeds TRNG

#if SPI_ENABLED
    spi_bus_init();
#elif I2C_ENABLED
    i2c_bus_init();
#endif

if (!wallet.begin()) {
    ESP_LOGE(TAG, "PN532 init failed");
    vTaskDelay(portMAX_DELAY);
}
```

3. Open the channel, sign, and wipe:

```
CW_SecureSession session;
if (wallet.connect(session)) {
    uint8_t testHash[CW_HASH_SIZE];
    memset(testHash, 0x01, sizeof(testHash));   // replace with SHA-256(tx)

    CW_SignRequest req(session, CW_SIGN_CURR_K1,
                       CW_SIGN_SIG_ECDSA_LOW_S, CW_SIGN_WITH_PIN);
    req.hash       = testHash;
    req.hashLength = sizeof(testHash);
    (void)CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
                                DEFAULT_PIN, DEFAULT_PIN_LEN);

    CW_SignResult sig = wallet.sign(req);
    // sig.signature = r[32] || s[32], sig.errorCode == CW_OK on success

    CW_Utils::secure_wipe(testHash, sizeof(testHash));
    CW_Utils::secure_wipe(sig.signature, sizeof(sig.signature));
}
wallet.disconnect(session);
```

3.1.2.5 Hardening for production

This example is a demo. Before shipping firmware to end-users:

- **Silence the SDK logs.** The `ESP32Logger` writes through `ESP_LOG*`. Set `CONFIG_LOG_↩ DEFAULT_LEVEL_NONE=y` (or `esp_log_level_set("*", ESP_LOG_NONE)` at boot) to stop the SDK from emitting APDU and key material on UART0 in shipping firmware.
- **Move the PIN off flash.** A hardcoded `DEFAULT_PIN` lives in `.rodata` and is recoverable via JTAG / firmware dump. In production read the PIN at runtime (capacitive keypad, BLE prompt, secure element) and call `CW_Utils::secure_wipe` on the buffer after `wallet.sign()`.
- **Guard the PIN retry counter.** Apply the halt-on-wrong-PIN pattern from VerifyPin / Sign so the loop cannot exhaust the on-card counter.

3.1.2.6 Troubleshooting

Symptom	Cause	Fix
`Please enable exactly one of SPI_ENABLED / I2C_ENABLED` (build error)	Both macros are 0 or both are 1	Set exactly one to 1
`PN532 init failed`	Reader wiring / bus mode switches	Check VCC = 3.3 V, the switches and the configured pins — see `hardware setup`
Wi-Fi never connects	Wrong SSID / password, or 5 GHz-only network	ESP32-S3 is 2.4 GHz only — verify `main/config.h`
`Sign failed, error↩ Code: 0x81`	No seed on the card	`cryptnox seed generate`
`Sign failed, error↩ Code: 0x82`	Wrong PIN	Edit `DEFAULT_PIN`, re-flash — see VerifyPin

3.1.2.7 License

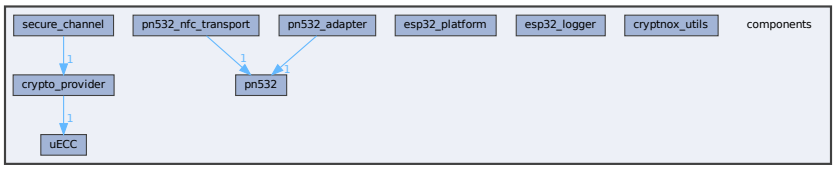
`cryptnox-sdk-esp32` is dual-licensed:

- **LGPL-3.0** for open-source projects and proprietary projects that comply with LGPL requirements
- **Commercial license** for projects that require a proprietary license without LGPL obligations (see COMMERCIAL.md for details)

For commercial inquiries, contact: `contact@cryptnox.com`

3.2 components Directory Reference

Directory dependency graph for components:

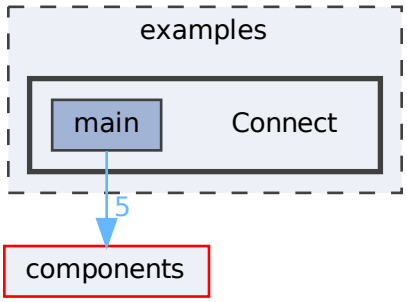


Directories

- directory cryptnox_utils
- directory crypto_provider
- directory esp32_logger
- directory esp32_platform
- directory pn532
- directory pn532_adapter
- directory pn532_nfc_transport
- directory secure_channel
- directory uECC

3.3 examples/Connect Directory Reference

Directory dependency graph for Connect:



Directories

- directory main

Files

- file config.template.h

3.3.1 Detailed Description

3.3.1.0.1 cryptnox-sdk-esp32

ESP32 SDK for managing Cryptnox smart card wallets

3.3.2 Connect — Secure Channel + Card Info

Open the Cryptnox Hardware Wallet secure channel from an ESP32-S3 and read back the card owner's **name** and **email**. This is the **safest starting point**: the firmware never sends a PIN, so it cannot lock the card.

3.3.2.1 Requirements

Component	Details
Hardware Wallet	Cryptnox Hardware Wallet, initialised (`cryptnox init`)
NFC reader	PN532 over SPI — MOSI=11, MISO=13, SCLK=12, CS=10 (see `hardware setup`)
Board	ESP32-S3-DevKitC-1
Toolchain	ESP-IDF v5.5
Repository	Cloned with submodules (`git clone --recurse-submodules ...`)

No PIN, no seed required. Wi-Fi credentials in `main/config.h` are still mandatory — the radio is started on boot so it feeds the ESP32 hardware TRNG with full entropy before any crypto runs.

3.3.2.2 Quick start

```
# from the repository root, after sourcing ESP-IDF env
cd examples/Connect
cp config.template.h main/config.h          # create your runtime config (gitignored)
$EDITOR main/config.h                       # fill WIFI_SSID / WIFI_PASSWORD
idf.py set-target esp32s3                   # once
idf.py build flash monitor
```

Exit the serial monitor with `Ctrl-]`. Place the card on the PN532 antenna once the firmware reports `PN532 ready`.

3.3.2.2.1 Expected output

```
I (520) wifi:wifi driver task: ...
I (1234) connect: Wi-Fi connected, TRNG seeded
I (1280) connect: PN532 firmware 1.6, features 0x07 (MIFARE + ISO-DEP + FeliCa)
I (2410) connect: Card connected, secure channel established
I (2420) connect: Owner name : Alice
I (2421) connect: Owner email: alice@cryptnox.com
```

3.3.2.3 How it works

```
wifi_init() + esp_wifi_connect()    Bring up Wi-Fi → TRNG ready
       |
spi_bus_initialize()                Configure SPI3 host
       |
wallet.begin()                      Reset the PN532, probe firmware
       |
wallet.connect(session)             SELECT + cert chain verify
       |                            (secp256r1) + ECDH + mutual auth
       |                            → session.aesKey / macKey / iv
       |
wallet.getCardInfo(session)         Secured APDU (AES-CBC + MAC)
       |                            Decrypt response, parse name & email
       |
wallet.disconnect(session)          Zero session keys
```

3.3.2.4 Step-by-step code

Wire the adapters together (declared once in `app_main`):

```
ESP32Logger          logger;
Pn532NfcTransport    nfc(/*pn532*/ &pn532, logger);
ESP32CryptoProvider  cryptoProvider;
ESP32Platform        platform;
CryptnoxWallet       wallet(nfc, logger, cryptoProvider, platform);
```

Bring up the radio + bus + reader:

```
ESP_ERROR_CHECK(nvs_flash_init());
wifi_init();                            // brings up Wi-Fi → seeds the HW TRNG
spi_bus_init();
if (!wallet.begin()) {
    ESP_LOGE(TAG, "PN532 init failed");
    vTaskDelay(portMAX_DELAY);
}
```

Open the channel and read the card in a tight loop:

```
while (true) {
    CW_SecureSession session;
    if (wallet.connect(session)) {
        CW_CardInfo info;
        if (wallet.getCardInfo(session, &info)) {
            ESP_LOGI(TAG, "Owner name : %s", info.name);
            ESP_LOGI(TAG, "Owner email: %s", info.email);
        }
    }
    wallet.disconnect(session);
    vTaskDelay(pdMS_TO_TICKS(1000));
}
```

3.3.2.5 Troubleshooting

Symptom	Cause	Fix
`PN532 init failed`	Reader wiring / power / mode switches	Check VCC = 3.3 V, the MOSI/MISO/SCLK/CS pinout and the PN532 SPI mode switches (`SW0=HIGH`, `SW1=LOW`) — see `hardware setup`
Wi-Fi never connects	Wrong SSID / password, or 5 GHz-only network	ESP32-S3 supports 2.4 GHz only — verify SSID, password, band in `main/config.h`
`Card not detected or secure channel failed`	Card off the antenna, or card not initialised	Bring the card within ~1 cm of the antenna; run `cryptnox init` if it is a brand-new card
`getCardInfo failed (channel error or parse error)`	Card initialised without an owner name/email	Re-run `cryptnox init` and fill in the owner fields

3.3.2.6 License

`cryptnox-sdk-esp32` is dual-licensed:

- **LGPL-3.0** for open-source projects and proprietary projects that comply with LGPL requirements
- **Commercial license** for projects that require a proprietary license without LGPL obligations (see COMMERCIAL.md for details)

For commercial inquiries, contact: `contact@cryptnox.com`

3.4 components/cryptnox_utils Directory Reference

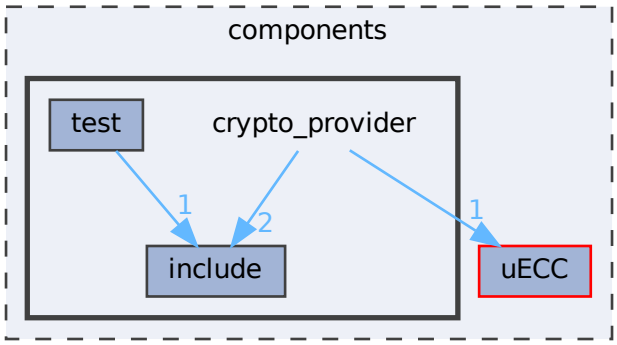
Directory dependency graph for cryptnox_utils:

Files

- file esp32_random.cpp

3.5 components/crypto_provider Directory Reference

Directory dependency graph for crypto_provider:



Directories

- directory include
- directory test

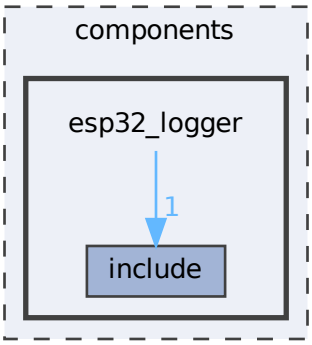
Files

- file esp32_crypto_provider.cpp

  Implementation of ESP32CryptoProvider — mbedTLS + hardware TRNG backend.

3.6 components/esp32_logger Directory Reference

Directory dependency graph for esp32_logger:



Directories

- directory include

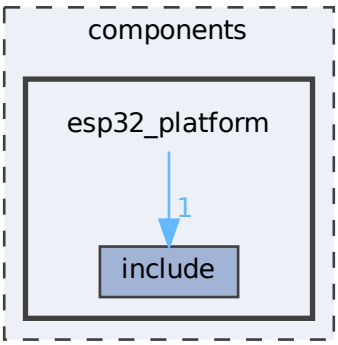
Files

- file ESP32Logger.cpp

  Implementation of ESP32Logger — ESP32 UART0 logging backend.

3.7 components/esp32_platform Directory Reference

Directory dependency graph for esp32_platform:



Directories

- directory include

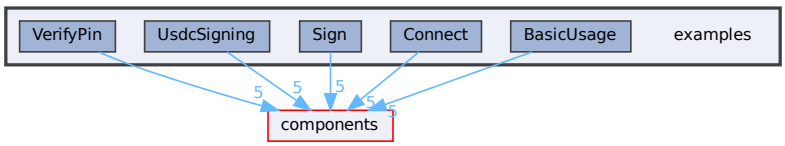
Files

- file ESP32Platform.cpp

  Implementation of ESP32Platform — FreeRTOS sleep backend.

3.8 examples Directory Reference

Directory dependency graph for examples:



Directories

- directory BasicUsage
- directory Connect
- directory Sign
- directory UsdcSigning
- directory VerifyPin

3.8.1 Detailed Description

3.8.1.0.1 cryptnox-sdk-esp32

ESP32 SDK for managing Cryptnox smart card wallets

3.8.2 Examples

Standalone ESP-IDF projects that exercise the Cryptnox Hardware Wallet over NFC (PN532). Each project ships with its own focused README so a reader landing on a single example gets everything needed end-to-end.

3.8.2.1 Prerequisites

Component	Details
Hardware Wallet	Cryptnox Hardware Wallet (firmware $\geq$ v1.6.0), initialised with a PIN — and a seed loaded for the signing examples
NFC reader	`PN532 NFC module` wired over **SPI** (default) or **I²C** — see `hardware setup`
Board	`ESP32-S3-DevKitC-1` (Espressif reference dev kit)
Toolchain	`ESP-IDF v5.5` installed and sourced (`. $IDF_PATH/export.sh`)

Component	Details
Repository	This repository **cloned with submodules** (`git clone --recurse-submodules ...`)

Provision a card from a host with a PC/SC reader and the Cryptnox CLI:

```
cryptnox init           # sets the PIN + PUK
cryptnox seed generate        # generates a BIP39 seed (required for signing)
```

Note

Every example starts Wi-Fi on boot so the radio feeds the ESP32 hardware TRNG with full entropy. Even the non-networked examples (`Connect`, `VerifyPin`, `Sign`, `BasicUsage`) need a valid `WIFI_SSID` / `WIFI_PASSWORD` in their `main/config.h`.

3.8.2.2 Available examples

Example	What it does
Connect	Opens the secure channel and reads back the card owner's name & email. **Safest first example** — no PIN, no signing, can't lock the card.
VerifyPin	Opens the secure channel and submits a PIN. Halts on a wrong PIN to protect the on-card retry counter.
Sign	Signs a 32-byte hash with the card's secp256k1 key. Returns the raw `r s` signature ready to broadcast.
BasicUsage	End-to-end walkthrough in one project: pick SPI **or** I^2C, open the channel, sign a hash. Good reference for production wiring.
UsdcSigning	Real-world flow: build an EIP-1559 USDC transfer, sign it on the card, broadcast it on Sepolia.

3.8.2.3 How to build and run an example

From the project root (where `idf.py`'s parent `CMakeLists.txt` lives):

```
. $IDF_PATH/export.sh                     # source ESP-IDF env (once per shell)
cd examples/<ExampleName>
cp main/config.h main/config.h.bak        # if you want to keep the template
$EDITOR main/config.h                     # fill WIFI_SSID / WIFI_PASSWORD (+ extras for UsdcSigning)
idf.py set-target esp32s3                 # once, writes sdkconfig
idf.py build flash monitor                # build, flash, open serial @ 115200 baud
```

Exit the monitor with `Ctrl-]`. Place the card on the PN532 antenna when the firmware prompts for it.

Note

All examples default to PIN `000000000` (nine zeros). If your card was initialised with a different PIN, edit the `DEFAULT_PIN` / `DEMO_PIN` / `CARD_PIN` macro at the top of the relevant source file before building.

3.8.2.4 Adding a new example

Follow the conventions used by the existing projects:

- Place each project in its own subdirectory under `examples/`, named in **PascalCase**.
- Lay it out as an ESP-IDF project: `CMakeLists.txt` at the project root, a `main/` subdirectory with `main/CMakeLists.txt` + sources, and a `sdkconfig.defaults` capturing the project's build options.
- Start every source file with the SPDX + copyright header used by the rest of the repository.
- Add Doxygen tags (`@file`, `@example`, `@brief`) so the project surfaces correctly in the generated docs.

- If the project needs external secrets (Wi-Fi credentials, RPC keys, recipient addresses), ship a `config.template.h` next to it and add the runtime `config.h` to `.gitignore`. Never commit credentials.
- Ship a `README.md` next to the project and register it in the Available examples table above.

3.8.2.5 License

`cryptnox-sdk-esp32` is dual-licensed:

- **LGPL-3.0** for open-source projects and proprietary projects that comply with LGPL requirements
- **Commercial license** for projects that require a proprietary license without LGPL obligations (see COMMERCIAL.md for details)

For commercial inquiries, contact: `contact@cryptnox.com`

3.9 components/crypto_provider/include Directory Reference

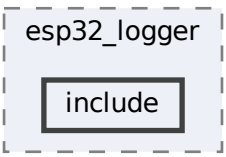
Directory dependency graph for include:

Files

- file esp32_crypto_provider.h
  CW_CryptoProvider implementation for ESP32 using mbedTLS and the hardware TRNG.

3.10 components/esp32_logger/include Directory Reference

Directory dependency graph for include:



Files

- file ESP32Logger.h

  CW_Logger implementation that writes to ESP32 UART0 via `printf`*.*

3.11 components/esp32_platform/include Directory Reference

Directory dependency graph for include:

Files

- file ESP32Platform.h

  CW_Platform implementation for ESP32 using FreeRTOS.

3.12 components/pn532/include Directory Reference

Directory dependency graph for include:

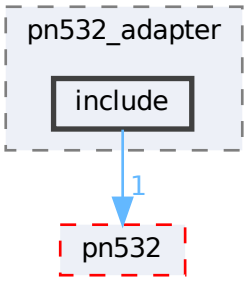
Files

- file pn532.h

  Low-level PN532 NFC controller driver for ESP-IDF (SPI and I^2C*).*

3.13 components/pn532_adapter/include Directory Reference

Directory dependency graph for include:



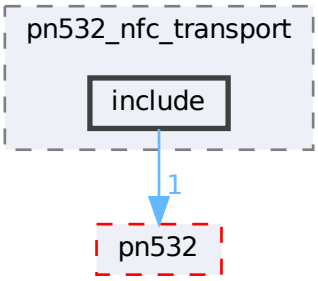
Files

- file pn532_adapter.h

  Self-contained CW_NfcTransport adapter that owns its pn532_t handle.

3.14 components/pn532_nfc_transport/include Directory Reference

Directory dependency graph for include:



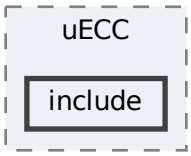
Files

- file Pn532NfcTransport.h

  CW_NfcTransport adapter wrapping the ESP-IDF PN532 NFC driver.

3.15 components/uECC/include Directory Reference

Directory dependency graph for include:

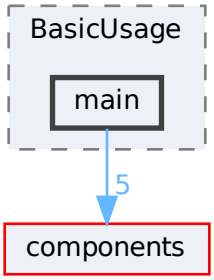


Files

- file uECC.h

3.16 examples/BasicUsage/main Directory Reference

Directory dependency graph for main:

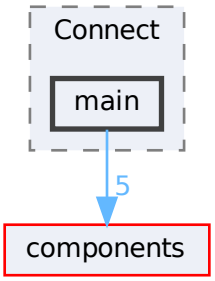


Files

- file main.cpp

3.17 examples/Connect/main Directory Reference

Directory dependency graph for main:

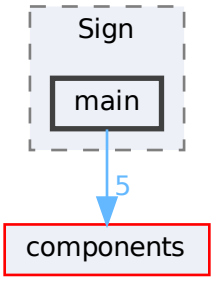


Files

- file main.cpp

3.18 examples/Sign/main Directory Reference

Directory dependency graph for main:

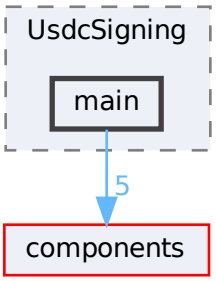


Files

- file main.cpp

3.19 examples/UsdcSigning/main Directory Reference

Directory dependency graph for main:

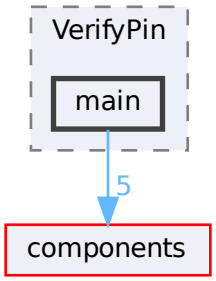


Files

- file eth_rlp.cpp
- file eth_rlp.h
- file eth_rpc.cpp
- file eth_rpc.h
- file keccak256.cpp
- file keccak256.h
- file main.cpp

3.20 examples/VerifyPin/main Directory Reference

Directory dependency graph for main:

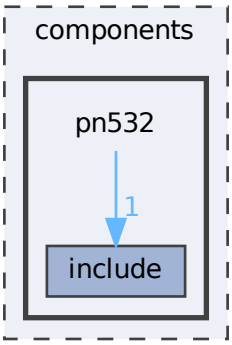


Files

- file main.cpp

3.21 components/pn532 Directory Reference

Directory dependency graph for pn532:



Directories

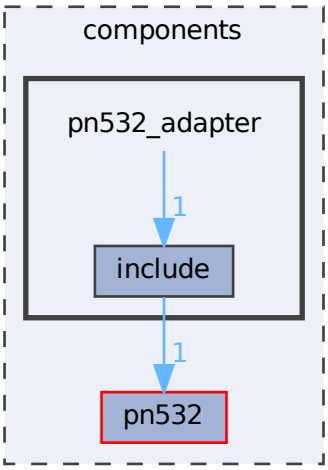
- directory include

Files

- file pn532.c

3.22 components/pn532_adapter Directory Reference

Directory dependency graph for pn532_adapter:



Directories

- directory include

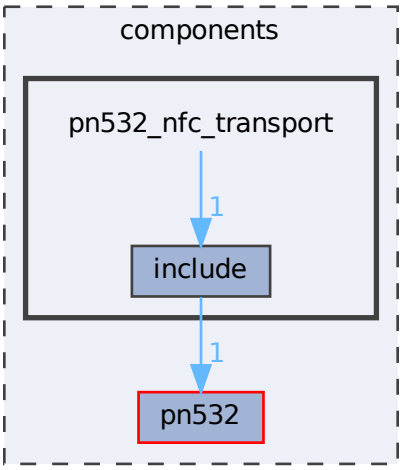
Files

- file pn532_adapter.cpp

  Implementation of PN532Adapter — self-contained PN532 NFC transport.

3.23 components/pn532_nfc_transport Directory Reference

Directory dependency graph for pn532_nfc_transport:



Directories

- directory include

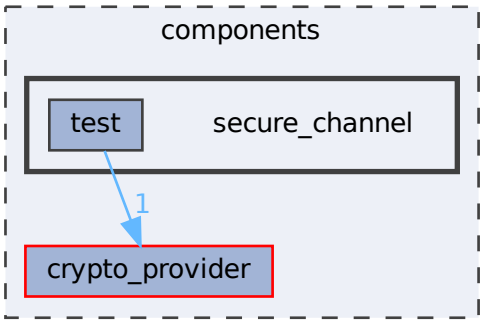
Files

- file Pn532NfcTransport.cpp

  Implementation of Pn532NfcTransport — ESP-IDF PN532 NFC transport.

3.24 components/secure_channel Directory Reference

Directory dependency graph for secure_channel:

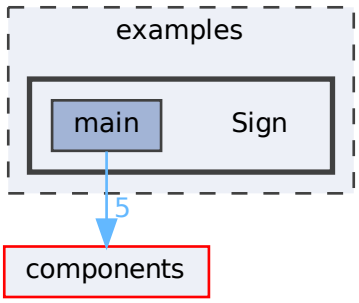


Directories

- directory test

3.25 examples/Sign Directory Reference

Directory dependency graph for Sign:



Directories

- directory main

Files

- file config.template.h

3.25.1 Detailed Description

3.25.1.0.1 cryptnox-sdk-esp32

ESP32 SDK for managing Cryptnox smart card wallets

3.25.2 Sign — ECDSA secp256k1 on a 32-Byte Hash

Sign an arbitrary 32-byte digest on the Cryptnox Hardware Wallet using the **secp256k1** curve (Bitcoin, Ethereum, BSC, Polygon, ...). The private key never leaves the card; the ESP32 only ever sees the hash and the resulting `(r, s)`.

Warning

> Every wrong PIN attempt decrements an on-card retry counter (typically 3–5 tries). At zero the PIN is permanently blocked. The firmware halts on `CW_SIGN_PIN_INCORRECT` (`0x82`) — do not remove the `vTaskDelay(portMAX_DELAY)` guard, and verify `DEMO_PIN` before flashing.

3.25.2.1 Requirements

Component	Details
Hardware Wallet	Cryptnox Hardware Wallet, initialised **and** seeded
NFC reader	PN532 over SPI — MOSI=11, MISO=13, SCLK=12, CS=10 (see `hardware setup`)
Board	ESP32-S3-DevKitC-1
Toolchain	ESP-IDF v5.5

`main/config.h` must contain valid `WIFI_SSID` / `WIFI_PASSWORD` — the radio is started on boot to seed the hardware TRNG before any crypto runs.
Provision the card from a host with a PC/SC reader and the `Cryptnox CLI`:

```
cryptnox init                # sets the PIN + PUK
cryptnox seed generate       # generates a BIP39 seed
```

Without a seed the SDK returns `CW_SIGN_NO_KEY_LOADED` (`0x81`).

3.25.2.2 Quick start

1. **Edit `DEMO_PIN`** in `main/main.cpp` to match your card's PIN.
2. **Create your runtime config from the template** (kept out of git so credentials never get committed) and fill in Wi-Fi:

   ```
   cp config.template.h main/config.h
   # then edit main/config.h: set WIFI_SSID / WIFI_PASSWORD
   ```

3. Build, flash and monitor:

   ```
   cd examples/Sign
   idf.py set-target esp32s3              # once
   idf.py build flash monitor
   ```

4. Exit the monitor with `Ctrl-]`. Place the card on the PN532 antenna.

3.25.2.2.1 Expected output

```
I (1280) sign: Card connected, secure channel established
I (1450) sign: Sign OK - r:
I (1450) sign: 7c 1f 3a 92 5e 0b 8c d4 92 0f ab 7c 53 e1 b9 d8 1f 2a 4c 76 e9 d8 05 bc 5d 2e ad 12 c3 fa 47
      cb
I (1450) sign: s:
I (1450) sign: 12 e0 bc 4f a7 88 09 67 da 1f 0e bd 83 c2 b7 91 58 0a c4 d6 2e 16 03 9f 7b 4c dd f1 3e 2a 89
      c5
```

`r` and `s` together form the 64-byte raw ECDSA signature (`signature[0..31]` = r, `signature[32..63]` = s). The card returns a **canonical low-S** signature (S $\leq$ n/2), so the output is directly forwardable to any chain that enforces BIP-62.

3.25.2.3 How it works

```
wallet.connect(session)        SELECT + cert verify + ECDH + mutual auth
       |
Build CW_SignRequest:
  keyType        = CW_SIGN_CURR_K1         (secp256k1, current key)
  signatureType  = CW_SIGN_SIG_ECDSA_LOW_S (canonical low-S, BIP-62)
  pinLessMode    = CW_SIGN_WITH_PIN        (PIN included in the payload)
  hash[32]       = your digest             (here a test pattern 0x01 × 32)
  pin[]          = DEMO_PIN
       |
wallet.sign(req)               Secured SIGN APDU. The card checks the
       |                       PIN, signs the hash with its secp256k1
       |                       private key, and replies with r  s.
       |
secure_wipe(hash, signature)   Zero local copies
       |
wallet.disconnect(session)     Zero session keys
```

3.25.2.3.1 Choosing the key path

`CW_SIGN_CURR_K1` signs with the card's **current** secp256k1 key (usually `m/`). To derive a BIP-44 sub-key first, set `derivePath` / `derivePathLength` on the request and use `CW_SIGN_DERIVE_K1` — see UsdcSigning for a worked example (`m/44'/60'/0'/0/0` for Ethereum).

3.25.2.3.2 PIN: in payload vs pre-verified

Mode	Round-trips	Notes
`CW_SIGN_WITH_PIN` (this example)	1	PIN included inside the SIGN APDU; one shot
`CW_SIGN_PINLESS` after `verifyPin()`	2	One PIN verification covers many subsequent signatures

Pre-verifying is preferable when you sign more than one hash per session.

3.25.2.4 Step-by-step code

Build the sign request:

```
uint8_t hash[CW_HASH_SIZE];
memset(hash, 0x01, sizeof(hash));        // replace with SHA-256 of your tx

CW_SignRequest req(session,
                   CW_SIGN_CURR_K1,
                   CW_SIGN_SIG_ECDSA_LOW_S,
                   CW_SIGN_WITH_PIN);
req.hash       = hash;
req.hashLength = sizeof(hash);
(void)CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
                            DEMO_PIN, CW_MAX_PIN_LENGTH);
```

Sign and handle the result:

```
CW_SignResult sig = wallet.sign(req);
if (sig.errorCode == CW_OK) {
    // sig.signature = r[32] || s[32] -- raw, ready to forward / DER-encode
} else if (sig.errorCode == CW_SIGN_PIN_INCORRECT) {
    CW_Utils::secure_wipe(hash, sizeof(hash));
    CW_Utils::secure_wipe(sig.signature, sizeof(sig.signature));
    wallet.disconnect(session);
    vTaskDelay(portMAX_DELAY);            // protect the retry counter
}
```

Wipe secrets before the next iteration:

```
CW_Utils::secure_wipe(hash, sizeof(hash));
CW_Utils::secure_wipe(sig.signature, sizeof(sig.signature));
wallet.disconnect(session);
```

`CW_Utils::secure_wipe` is a `volatile`-pointer memset that the compiler cannot elide — required to keep secrets from lingering in RAM.

3.25.2.5 Error codes

errorCode	Meaning	Action
`CW_OK` (`0x00`)	Signature OK	Use `sig.signature`
`CW_SIGN_NO_KEY_LOADED` (`0x81`)	Card has no seed	`cryptnox seed generate`
`CW_SIGN_PIN_INCORRECT` (`0x82`)	Wrong PIN	Halt — fix `DEMO_PIN` before re-running
other	Channel error / unexpected SW	Check the raw status word printed by the SDK

3.25.2.6 Troubleshooting

Symptom	Cause	Fix
`Sign failed: 0x81`	No seed on card	`cryptnox seed generate`
`Sign failed: 0x82`	Wrong PIN	Edit `DEMO_PIN`, re-flash — do **not** keep retrying
`APDU exchange failed!`	NFC link dropped mid-exchange	Hold the card steady through the LED pulse
`Card not detected`	Card not on the antenna	Bring the card within ~1 cm of the antenna

3.25.2.7 License

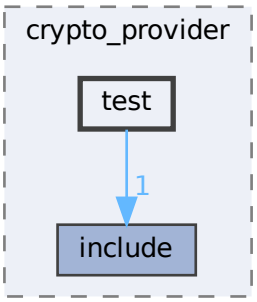
`cryptnox-sdk-esp32` is dual-licensed:

- **LGPL-3.0** for open-source projects and proprietary projects that comply with LGPL requirements
- **Commercial license** for projects that require a proprietary license without LGPL obligations (see COMMERCIAL.md for details)

For commercial inquiries, contact: `contact@cryptnox.com`

3.26 components/crypto_provider/test Directory Reference

Directory dependency graph for test:

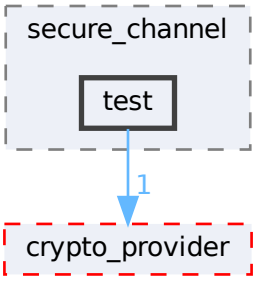


Files

- file test_esp32_crypto_provider.cpp

3.27 components/secure_channel/test Directory Reference

Directory dependency graph for test:

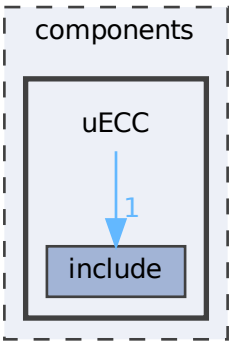


Files

- file test_cw_secure_channel.cpp

3.28 components/uECC Directory Reference

Directory dependency graph for uECC:



Directories

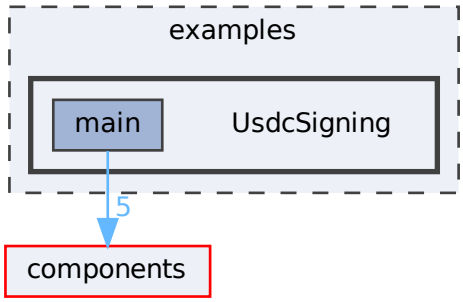
- directory include

Files

- file uECC_esp32.cpp

3.29 examples/UsdcSigning Directory Reference

Directory dependency graph for UsdcSigning:



Directories

- directory main

Files

- file config.template.h

3.29.1 Detailed Description

3.29.1.0.1 cryptnox-sdk-esp32

ESP32 SDK for managing Cryptnox smart card wallets

3.29.2 UsdcSigning — Broadcast a Real EIP-1559 USDC Transfer on Sepolia

End-to-end demonstration of the Cryptnox Hardware Wallet on an ESP32-S3: connect to a Wi-Fi access point, fetch the nonce and fee parameters over JSON-RPC, build and Keccak-256-hash the unsigned EIP-1559 transaction, sign it on the card (BIP-44 `m/44'/60'/0'/0/0`, secp256k1, canonical low-S), recover `yParity`, broadcast the signed transaction, and print the on-chain tx hash.
The private key never leaves the card. The ESP32 only ever holds the `Account` derived from the card's public key, the unsigned tx hash, and the resulting `(r, s)`.

Warning

Every wrong PIN attempt decrements an on-card retry counter. At zero the PIN is permanently blocked. The firmware halts on `CW_SIGN_PIN_INCORRECT` — verify `CARD_PIN` in `main/config.h` matches the value used during `cryptnox init` **before** flashing.

3.29.2.1 Requirements

Component	Details
Hardware Wallet	Cryptnox Hardware Wallet, initialised **and** seeded
Wallet funding	The address derived from `m/44'/60'/0'/0/0` on the card needs Sepolia ETH (for gas) and Sepolia USDC. Faucets: `Sepolia ETH` `Circle USDC`
NFC reader	PN532 over **SPI** or **I²C** — selected by `SPI_ENABLED` / `I2C_ENABLED` at the top of `main.cpp`; see `hardware setup`
Board	ESP32-S3-DevKitC-1 (or any ESP32 family — Wi-Fi required)
Toolchain	ESP-IDF v5.5
RPC endpoint	A Sepolia JSON-RPC endpoint — `PublicNode` (no signup, default) or `Infura`

3.29.2.2 Quick start

1. **Create `main/config.h`** by copying the template:
   ```
   cp config.template.h main/config.h
   ```
   Fill in at minimum: `WIFI_SSID`, `WIFI_PASSWORD`, `CARD_PIN`, `ADDR_FROM` (the address derived from `m/44'/60'/0'/0/0` on the card), `ADDR_TO` (recipient), `AMOUNT_USDC` (token base units — 1 USDC = 1 000 000).

Important

`main/config.h` is gitignored — never commit it.

1. Build, flash and monitor:
   ```
   cd examples/UsdcSigning
   idf.py set-target esp32s3              # once
   idf.py build flash monitor
   ```
2. Exit the monitor with `Ctrl-]`. Place the card on the PN532 antenna when the firmware prompts for it.

3.29.2.2.1 Expected output

```
I (1234) usdc_signing: Wi-Fi connected, TRNG seeded
I (1290) usdc_signing: PN532 ready
I (1340) usdc_signing: fetchNonce: status=200 nonce=0x5
I (1342) usdc_signing: [HASH]: 0x6679a2cd3064046397addbb97004b606df9281f624409fd36d2d24832db59c29
I (1500) usdc_signing: Card connected, secure channel established
I (1740) usdc_signing: [SIG r]: 7C1F3A925E0B8CD4920FAB7C53E1B9D81F2A4C76E9D805BC5D2EAD12C3FA47CB
I (1741) usdc_signing: [SIG s]: 12E0BC4FA7880967DA1F0EBD83C2B791580AC4D62E16039F7B4CDDF13E2A89C5
I (2030) usdc_signing: yParity: 1
I (2031) usdc_signing: Sending...
I (2350) usdc_signing: [RPC] HTTP 200
I (2351) usdc_signing: [tx] hash=0xab12cd34ef56...
I (2351) usdc_signing: Transaction sent successfully!
```

Paste the final `[tx] hash` into `Sepolia Etherscan` to watch confirmation.

3.29.2.3 How it works

```
 1. wifi_init() + esp_wifi_connect()
        |
 2. eth_rpc_fetch_nonce()              POST eth_getTransactionCount
        |                              over TLS via esp_http_client
        |
 3. Build Tx2 struct                   chainId=11155111, nonce, fees,
        |                              gasLimit, to, value=0, data
        |
 4. eth_rlp_encode_erc20_transfer()    0xa9059cbb || pad(to,32) ||
        |                              pad(AMOUNT_USDC,32)
        |
 5. eth_rlp_encode_unsigned_tx()       0x02 || rlp([chainId, nonce,
        |                              ..., [] accessList])
        |
 6. keccak256(rlp, len, hash)          EIP-2718 typed-tx pre-image (32 B)
        |
 7. wallet.connect(session)            Secure channel
        |
 8. Build CW_SignRequest:
      keyType    = CW_SIGN_DERIVE_K1    (BIP-44 m/44'/60'/0'/0/0)
      sigType    = CW_SIGN_SIG_ECDSA_LOW_S
      pinMode    = CW_SIGN_WITH_PIN
      hash       = keccak256(unsigned tx)
      pin        = CARD_PIN
        |
 9. wallet.sign(req)                   64-byte r || s
        |
10. eth_rpc_ecrecover_parity(...)      Call the ecrecover precompile
        |                              with v=27, then v=28 -- keep the
        |                              one that returns ADDR_FROM
        |
11. eth_rlp_encode_signed_tx()         Same RLP plus the signature fields
        |
12. eth_rpc_send_raw_tx()              eth_sendRawTransaction → extract
        |                              "result":"0x..." tx hash
        |
13. secure_wipe(req.pin, ...)          Zero the PIN buffer
```

3.29.2.3.1 TLS

The ESP-IDF HTTP client uses the bundled certificate bundle (set via `CONFIG_MBEDTLS_↩ CERTIFICATE_BUNDLE=y` in `sdkconfig.defaults`) so the RPC connection is validated against the same root store ESP-IDF applications use by default. No per-provider PEM is required as long as the endpoint chains to one of the included roots.

3.29.2.3.2 `yParity` recovery

EIP-1559 signatures carry a 1-bit `yParity` instead of EIP-155's `v`. The card returns only `r` and `s`, so the firmware calls the `ecrecover precompile` at address `0x01` with `v = 27` (yParity = 0). If the recovered address matches `ADDR_FROM` the firmware keeps yParity = 0; otherwise it retries with `v = 28` (yParity = 1). One of the two will match when the card's key and `ADDR_FROM` are consistent.

3.29.2.4 Configuration reference

All fields live in `main/config.h` (gitignored). Start from `config.template.h` and fill in:

3.29.2.4.1 Wi-Fi

Field	Required	Example
WIFI_SSID	yes	"MyHomeNetwork"
WIFI_PASSWORD	yes	"password"

Note

The ESP32-S3 (and every Wi-Fi-capable ESP32 family member) supports **2.4 GHz** Wi-Fi only.

3.29.2.4.2 RPC endpoint — choose one provider

PublicNode (free, no signup, default):

```
#define RPC_URL    "https://ethereum-sepolia-rpc.publicnode.com"
```

Infura (free tier, requires API key):

```
#define RPC_URL    "https://sepolia.infura.io/v3/<YOUR_INFURA_PROJECT_ID>"
```

3.29.2.4.3 Wallet

Field	Example
CARD_PIN	"000000000" (must match cryptnox init)
ADDR_FROM	"<40 lowercase hex chars>" — your card's Ethereum address (no 0x prefix)

ADDR_FROM must equal the address derived from m/44'/60'/0'/0/0 on the card. If they disagree, the yParity recovery loop cannot find a matching value and the firmware halts.

3.29.2.4.4 Transaction

Field	Default	Notes
ADDR_TO	"Cd7E5...c06e"	Recipient address (hex, no 0x)
ADDR_USDC	"1c7D4...7238"	USDC contract on Sepolia
CHAIN_ID_SEPOLIA	11155111	Hardcoded — no path to broadcast on mainnet by accident
AMOUNT_USDC	1000000UL	1 USDC (6 decimals)
MAX_PRIORITY_FEE	2000000000ULL	2 Gwei
MAX_FEE	4000000000ULL	4 Gwei
GAS_LIMIT_ERC20	60000ULL	Standard ERC-20 transfer

3.29.2.5 Memory footprint

After idf.py build:

Section	Size
Flash (.text + .rodata)	~900 KB
RAM at runtime	~80 KB

The bulk of the flash is the IDF Wi-Fi stack, mbedTLS, and the bundled CA store (~250 KB). The Cryptnox SDK itself plus the example's eth_rlp / eth_rpc / keccak256 helpers sit around ~50 KB.

3.29.2.6 Troubleshooting

Symptom	Cause	Fix
`esp-tls: mbedtls_↩ ssl_handshake returned -0x2700`	RPC's chain not in the bundled root store	Switch to PublicNode (Google Trust Services R4 is bundled) or add the custom root via the certificate-bundle component
Wi-Fi never connects	Wrong SSID / password, or a 5 GHz-only network	ESP32-S3 is 2.4 GHz only — verify `WIFI_SSID` / `WIFI_PASSWORD`
`tx.to is not a valid 40-char hex string`	Bad hex in `ADDR_TO`	40 hex chars, no `0x`, lowercase preferred
`yParity determination failed!`	`ADDR_FROM` doesn't match the card's m/44'/60'/0'/0/0 address	Derive the address with the `Cryptnox CLI` and copy it into `ADDR_FROM`
RPC returns `nonce too low` in the JSON `error.message`	Stale nonce from the RPC	Wait a few seconds and re-run; the firmware fetches the nonce just before signing
`Wrong PIN -- halting to protect retry counter`	`CARD_PIN` does not match the card	Fix `CARD_PIN` — do **not** keep retrying, every attempt burns one of the on-card tries (see VerifyPin)
`Sign failed: 0x81`	Card has no seed	`cryptnox seed generate`

3.29.2.7 License

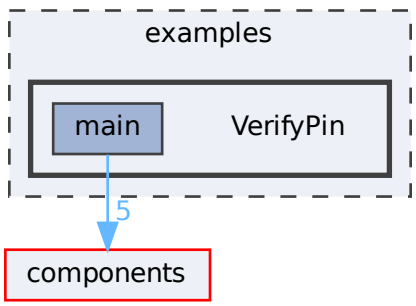
`cryptnox-sdk-esp32` is dual-licensed:

- **LGPL-3.0** for open-source projects and proprietary projects that comply with LGPL requirements
- **Commercial license** for projects that require a proprietary license without LGPL obligations (see COMMERCIAL.md for details)

For commercial inquiries, contact: `contact@cryptnox.com`

3.30 examples/VerifyPin Directory Reference

Directory dependency graph for VerifyPin:



Directories

- directory main

Files

- file config.template.h

3.30.1 Detailed Description

3.30.1.0.1 cryptnox-sdk-esp32

ESP32 SDK for managing Cryptnox smart card wallets

3.30.2 VerifyPin — PIN Verification Over the Secure Channel

Open the Cryptnox Hardware Wallet secure channel and submit a PIN. The firmware prints `PIN accepted` on success and **halts** the loop on a wrong PIN so it cannot exhaust the on-card retry counter.

Warning

> **Read this before flashing.** Every wrong PIN attempt decrements an on-card retry counter (typically 3–5 tries). At zero the PIN is permanently blocked; recovery then requires the PUK via `cryptnox unblock_pin`. This example halts on a wrong PIN — do not remove the `vTaskDelay(portMAX_DELAY)` guard, and verify `DEMO_PIN` matches the value used during `cryptnox init` **before** flashing.

3.30.2.1 Requirements

Component	Details
Hardware Wallet	Cryptnox Hardware Wallet, initialised with a known PIN
NFC reader	PN532 over SPI — MOSI=11, MISO=13, SCLK=12, CS=10 (see `hardware setup`)
Board	ESP32-S3-DevKitC-1
Toolchain	ESP-IDF v5.5

`main/config.h` must contain valid `WIFI_SSID` / `WIFI_PASSWORD` — the radio is started on boot to seed the hardware TRNG before any crypto runs.

3.30.2.2 Quick start

1. **Edit `DEMO_PIN`** in `main/main.cpp` to match your card's PIN.
2. **Create your runtime config from the template** (kept out of git so credentials never get committed) and fill in Wi-Fi:

```
cp config.template.h main/config.h
# then edit main/config.h: set WIFI_SSID / WIFI_PASSWORD
```

3. Build, flash and monitor:

```
cd examples/VerifyPin
idf.py set-target esp32s3                # once
idf.py build flash monitor
```

4. Exit the monitor with `Ctrl-]`. Place the card on the PN532 antenna.

3.30.2.2.1 Expected output

Happy path (every second):

```
I (1280) verify_pin: Card connected, secure channel established
I (1305) verify_pin: PIN accepted
```

Wrong PIN (firmware then halts):

```
E (1280) verify_pin: Secured APDU: bad SW 0x63C2
E (1281) verify_pin: PIN rejected -- halting to protect retry counter
```

The SDK prints the raw `SW1 SW2` bytes on PIN failure:

- `0x63CN` — wrong PIN, **N** retries remaining
- `0x6983` — PIN already blocked (0 retries)

3.30.2.3 How it works

```
wallet.connect(session)            SELECT + cert verify + ECDH + mutual auth
       |
wallet.verifyPin(session, ...)     Secured APDU (AES-CBC + MAC) carrying
       |                           the PIN. Card checks it locally and
       |                           replies with SW 0x9000 on success or
       |                           0x63CN on wrong PIN with N retries left.
       |
wallet.disconnect(session)         Zero session keys
```

`verifyPin` returns `true` **only** when the card replies `0x9000`.

3.30.2.4 Step-by-step code

```
/* Replace with the PIN set on the card (4-9 ASCII digits). */
static const uint8_t DEMO_PIN[CW_MAX_PIN_LENGTH] = {
    '0', '0', '0', '0', '0', '0', '0', '0', '0'
};

CW_SecureSession session{};
if (!wallet.connect(session)) {
    ESP_LOGW(TAG, "Card not detected");
    wallet.disconnect(session);
    vTaskDelay(pdMS_TO_TICKS(1000));
    continue;
}

if (!wallet.verifyPin(session,
                      DEMO_PIN,
                      static_cast<uint8_t>(CW_MAX_PIN_LENGTH))) {
    ESP_LOGE(TAG, "PIN rejected -- halting to protect retry counter");
    wallet.disconnect(session);
    vTaskDelay(portMAX_DELAY);          // CRITICAL: do NOT loop on failure
}

ESP_LOGI(TAG, "PIN accepted");
wallet.disconnect(session);
```

3.30.2.5 Recovering a blocked PIN

If the PIN counter has reached zero, run the `Cryptnox CLI` on a host with a PC/SC reader:

```
cryptnox unblock_pin
```

The CLI prompts for the **PUK** (set during `cryptnox init`). Without the PUK the PIN cannot be reset and the on-card signing key material is unrecoverable.

3.30.2.6 Troubleshooting

Symptom	Cause	Fix
`PN532 init failed`	Reader wiring / SPI mode switches	See `hardware setup`

Symptom	Cause	Fix
Wi-Fi never connects	Wrong SSID / password, or 5 GHz-only network	ESP32-S3 is 2.4 GHz only — verify `WIFI_SSID` / `WIFI_PASSWORD` in `main/config.h`
`Card not detected`	Card not on the antenna	Bring the card within ~1 cm of the antenna
`bad SW 0x63CN`	Wrong PIN, N retries remaining	Fix `DEMO_PIN`, re-flash
`bad SW 0x6983`	PIN blocked (0 retries)	Run `cryptnox unblock_pin` with the PUK
`mutuallyAuthenticate failed`	Non-Cryptnox card or seed mismatch	Verify the card was set up with `cryptnox init`

3.30.2.7 License

`cryptnox-sdk-esp32` is dual-licensed:

- **LGPL-3.0** for open-source projects and proprietary projects that comply with LGPL requirements
- **Commercial license** for projects that require a proprietary license without LGPL obligations (see COMMERCIAL.md for details)

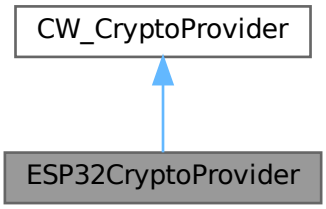
For commercial inquiries, contact: `contact@cryptnox.com`

Chapter 4

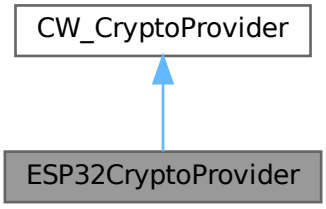
Class Documentation

4.1 ESP32CryptoProvider Class Reference

CW_CryptoProvider backed by mbedTLS and the ESP32 hardware TRNG.
`#include <esp32_crypto_provider.h>`
Inheritance diagram for ESP32CryptoProvider:



Collaboration diagram for ESP32CryptoProvider:



Public Member Functions

- bool sha256 (const uint8_t *data, size_t len, uint8_t *out) override
  Compute SHA-256 over a contiguous buffer.
- bool sha512 (const uint8_t *data, size_t len, uint8_t *out) override

Compute SHA-512 over a contiguous buffer.

- uint16_t aesCbcEncrypt (const uint8_t *in, uint16_t len, uint8_t *out, const uint8_t *key, uint8_t keyLen, uint8_t *iv, bool bitPadding) override

  Encrypt a buffer with AES-CBC.
- uint16_t aesCbcDecrypt (uint8_t *in, uint16_t len, uint8_t *out, const uint8_t *key, uint8_t key↩ Len, uint8_t *iv, bool bitPadding) override

  Decrypt a buffer with AES-CBC.
- bool ecdh (const uint8_t *pubKey, const uint8_t *privKey, uint8_t *secret, CW_Curve curve) override

  Compute an ECDH shared secret.
- bool makeKey (uint8_t *pubKey, uint8_t *privKey, CW_Curve curve) override

  Generate an ephemeral EC key pair.
- bool random (uint8_t *dest, unsigned size) override

  Fill a buffer with cryptographically random bytes.
- bool ecdsaVerify (const uint8_t *pubKey64, const uint8_t *hash, size_t hashLen, const uint8_t *sig, CW_Curve curve) override

  Verify an ECDSA signature.
- ~ESP32CryptoProvider () override

  Default destructor.

4.1.1 Detailed Description

CW_CryptoProvider backed by mbedTLS and the ESP32 hardware TRNG.
All operations are stateless; a single instance may be shared across multiple CryptnoxWallet objects (though concurrent use from different RTOS tasks is not thread-safe without external locking).

Warning

See the random method for TRNG entropy requirements (SEC-001).

Example

```
ESP32CryptoProvider crypto;
CW_CryptoProvider &provider = crypto;

uint8_t pub[64], priv[32];
provider.makeKey(pub, priv, CW_CURVE_SECP256R1);   // generate ephemeral key pair

uint8_t digest[32];
provider.sha256(message, messageLen, digest);       // hash a message
```

See also

CW_CryptoProvider

CryptnoxWallet

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 62 of file esp32_crypto_provider.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 ~ESP32CryptoProvider()

```
ESP32CryptoProvider::~ESP32CryptoProvider ()  [inline], [override]
```

Default destructor.
Definition at line 213 of file esp32_crypto_provider.h.

4.1.3 Member Function Documentation

4.1.3.1 aesCbcDecrypt()

```
uint16_t ESP32CryptoProvider::aesCbcDecrypt (
            uint8_t * in,
            uint16_t len,
            uint8_t * out,
            const uint8_t * key,
            uint8_t keyLen,
            uint8_t * iv,
            bool bitPadding)  [override]
```

Decrypt a buffer with AES-CBC.
AES-CBC decrypt with optional ISO/IEC 9797-1 Method 2 bit-padding removal.
Uses mbedTLS AES-CBC and optionally strips ISO/IEC 7816-4 bit-padding after decryption. The IV is updated in-place.

Parameters

in,out	*in*	Ciphertext input buffer; may be modified in-place (must not be NULL).
in	*len*	Length of in in bytes (must be a multiple of 16).
out	*out*	Plaintext output buffer (must not be NULL).
in	*key*	AES key bytes (must not be NULL).
in	*keyLen*	Key length in bytes (16 / 24 / 32).
in,out	*iv*	16-byte IV; updated in-place after the call.
in	*bitPadding*	When true, strips ISO/IEC 7816-4 bit padding from the decrypted output.

Returns

Number of plaintext bytes written to out, or 0 on failure.

Definition at line 147 of file esp32_crypto_provider.cpp.
References AES_BLOCK_SIZE_BYTES, AES_KEY_BITS_PER_BYTE, BIT_PADDING_MARKER, MBEDTLS_OK, and PADDING_ZERO_FILL.

4.1.3.2 aesCbcEncrypt()

```
uint16_t ESP32CryptoProvider::aesCbcEncrypt (
            const uint8_t * in,
            uint16_t len,
            uint8_t * out,
            const uint8_t * key,
            uint8_t keyLen,
            uint8_t * iv,
            bool bitPadding)  [override]
```

Encrypt a buffer with AES-CBC.
AES-CBC encrypt with optional ISO/IEC 9797-1 Method 2 bit padding.
Uses mbedTLS AES-CBC with optional ISO/IEC 7816-4 bit-padding. The IV is updated in-place after each block so that the caller can chain calls for streaming encryption.

Parameters

in	*in*	Plaintext input buffer (must not be NULL).
in	*len*	Length of in in bytes.
out	*out*	Ciphertext output buffer; must be at least len + 16 bytes to accommodate padding.

in	*key*	AES key bytes (must not be NULL).
in	*keyLen*	Key length in bytes (16 for AES-128, 24 for AES-192, 32 for AES-256).
in,out	*iv*	16-byte IV; updated in-place after the call.
in	*bitPadding*	When true, applies ISO/IEC 7816-4 bit padding before encrypting.

Returns

Number of ciphertext bytes written to out, or 0 on failure.

Definition at line 89 of file esp32_crypto_provider.cpp.
References AES_BLOCK_SIZE_BYTES, AES_KEY_BITS_PER_BYTE, AES_PAD_BUF_SIZE, BIT_PADDING_MARKER, MBEDTLS_OK, and PADDING_ZERO_FILL.

4.1.3.3 ecdh()

```
bool ESP32CryptoProvider::ecdh (
            const uint8_t * pubKey,
            const uint8_t * privKey,
            uint8_t * secret,
            CW_Curve curve)  [override]
```

Compute an ECDH shared secret.
Compute ECDH shared secret: X-coordinate of privKey * pubKey point.
Performs the standard Diffie-Hellman point multiplication secret = privKey × pubKey on the specified curve via the uECC shim.

Parameters

in	*pubKey*	Uncompressed peer public key (64 bytes, X||Y; no 0x04 prefix; must not be NULL).
in	*privKey*	32-byte private scalar (must not be NULL).
out	*secret*	32-byte shared-secret output buffer (must not be NULL).
in	*curve*	Elliptic curve selector (CW_CURVE_SECP256R1 or CW_CURVE_SECP256K1).

Returns

true on success, false if point multiplication fails or a pointer argument is NULL.

Definition at line 211 of file esp32_crypto_provider.cpp.
References toCurve(), uECC_shared_secret(), and UECC_SUCCESS.

4.1.3.4 ecdsaVerify()

```
bool ESP32CryptoProvider::ecdsaVerify (
            const uint8_t * pubKey64,
            const uint8_t * hash,
            size_t hashLen,
            const uint8_t * sig,
            CW_Curve curve)  [override]
```

Verify an ECDSA signature.
Verify an ECDSA signature (raw r||s) against a hash on the specified curve.
Checks that sig is a valid low-S DER-encoded ECDSA signature over hash produced by the private key corresponding to pubKey64.

Parameters

in	*pubKey64*	64-byte uncompressed public key (X||Y, no 0x04 prefix; must not be NULL).

in	*hash*	Message digest to verify against (must not be NULL).
in	*hashLen*	Length of hash in bytes.
in	*sig*	DER-encoded signature bytes (must not be NULL).
in	*curve*	Elliptic curve selector (CW_CURVE_SECP256R1 or CW_CURVE_SECP256K1).

Returns

true if the signature is valid, false otherwise (including on NULL arguments or malformed DER).

Definition at line 250 of file esp32_crypto_provider.cpp.
References toCurve(), UECC_SUCCESS, and uECC_verify().

4.1.3.5 makeKey()

```
bool ESP32CryptoProvider::makeKey (
            uint8_t * pubKey,
            uint8_t * privKey,
            CW_Curve curve)  [override]
```

Generate an ephemeral EC key pair.
Generate an ECC key pair via mbedTLS and the ESP32 hardware RNG.
Uses the hardware TRNG (via random) as the entropy source for the private scalar.

Parameters

out	*pubKey*	64-byte uncompressed public key output (X‖Y, no 0x04 prefix; must not be NULL).
out	*privKey*	32-byte private key output (must not be NULL).
in	*curve*	Elliptic curve selector (CW_CURVE_SECP256R1 or CW_CURVE_SECP256K1).

Returns

true on success, false on RNG or key-generation failure.

Warning

Ensure Wi-Fi or BT is active before calling this to guarantee full TRNG entropy (SEC-001).

Definition at line 223 of file esp32_crypto_provider.cpp.
References toCurve(), uECC_make_key(), and UECC_SUCCESS.

4.1.3.6 random()

```
bool ESP32CryptoProvider::random (
            uint8_t * dest,
            unsigned size)  [override]
```

Fill a buffer with cryptographically random bytes.
Fill dest with size cryptographically random bytes from the ESP32 hardware RNG.
Calls esp_fill_random() which reads from the ESP32 hardware TRNG. Full entropy requires Wi-Fi or Bluetooth to be active; without a live radio the TRNG operates in reduced-entropy mode (thermal noise and ring-oscillator jitter only — see SEC-001).

Parameters

out	*dest*	Buffer to fill (must not be NULL).

in	*size*	Number of random bytes to generate.

Returns

`true` on success, `false` if `dest` is `NULL` or `size` is `0`.

Warning

In production firmware, call this only after Wi-Fi or BT has been started to ensure full entropy.

Definition at line 239 of file esp32_crypto_provider.cpp.

4.1.3.7 sha256()

```
bool ESP32CryptoProvider::sha256 (
            const uint8_t * data,
            size_t len,
            uint8_t * out)  [override]
```

Compute SHA-256 over a contiguous buffer.
Compute SHA-256 over the input buffer, writing 32 bytes to out.
Uses the mbedTLS `mbedtls_sha256` API, which is hardware-accelerated on ESP32-S3.

Parameters

in	*data*	Pointer to the input data (must not be `NULL`).
in	*len*	Length of `data` in bytes.
out	*out*	32-byte output buffer for the digest (must not be `NULL`).

Returns

`true` on success, `false` if `data` or `out` is `NULL` or if the mbedTLS call fails.

Definition at line 73 of file esp32_crypto_provider.cpp.
References MBEDTLS_OK, and MBEDTLS_SHA256_MODE.

4.1.3.8 sha512()

```
bool ESP32CryptoProvider::sha512 (
            const uint8_t * data,
            size_t len,
            uint8_t * out)  [override]
```

Compute SHA-512 over a contiguous buffer.
Compute SHA-512 over the input buffer, writing 64 bytes to out.
Uses the mbedTLS `mbedtls_sha512` API, which is hardware-accelerated on ESP32-S3.

Parameters

in	*data*	Pointer to the input data (must not be `NULL`).
in	*len*	Length of `data` in bytes.
out	*out*	64-byte output buffer for the digest (must not be `NULL`).

Returns

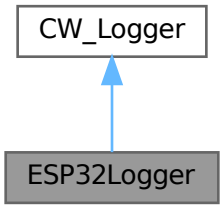
`true` on success, `false` if `data` or `out` is `NULL` or if the mbedTLS call fails.

Definition at line 79 of file esp32_crypto_provider.cpp.
References MBEDTLS_OK, and MBEDTLS_SHA512_MODE.
The documentation for this class was generated from the following files:

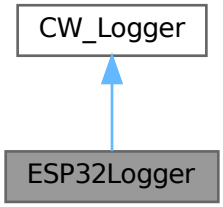
- components/crypto_provider/include/esp32_crypto_provider.h
- components/crypto_provider/esp32_crypto_provider.cpp

4.2 ESP32Logger Class Reference

CW_Logger backed by ESP32 UART0.
`#include <ESP32Logger.h>`
Inheritance diagram for ESP32Logger:



Collaboration diagram for ESP32Logger:



Public Member Functions

- ~ESP32Logger () override
  Default destructor.

Initialisation

- bool begin (unsigned long baudRate=115200UL) override
  Initialise UART0 at the given baud rate.

Print (no newline)

- void print (const __FlashStringHelper *str) override
  Print a PROGMEM string.
- void print (const char *str) override
  Print a NUL-terminated C string.
- void print (char c) override
  Print a single character.
- void print (uint8_t value, int base=DEC) override
  Print an 8-bit unsigned integer.
- void print (uint16_t value, int base=DEC) override
  Print a 16-bit unsigned integer.
- void print (uint32_t value, int base=DEC) override
  Print a 32-bit unsigned integer.
- void print (int value, int base=DEC) override
  Print a signed integer.

Println (with trailing newline)

- void println () override
  Print a CR+LF newline sequence.
- void println (const __FlashStringHelper *str) override
  Print a PROGMEM string followed by a newline.
- void println (const char *str) override
  Print a NUL-terminated C string followed by a newline.
- void println (char c) override
  Print a single character followed by a newline.
- void println (uint8_t value, int base=DEC) override
  Print an 8-bit unsigned integer followed by a newline.
- void println (uint16_t value, int base=DEC) override
  Print a 16-bit unsigned integer followed by a newline.
- void println (uint32_t value, int base=DEC) override
  Print a 32-bit unsigned integer followed by a newline.
- void println (int value, int base=DEC) override
  Print a signed integer followed by a newline.

Private Attributes

- bool m_initialized = false
  `true` *after a successful begin call.*

4.2.1 Detailed Description

CW_Logger backed by ESP32 UART0.
Outputs all log traffic through UART0 (the default USB-serial console on most ESP32 dev kits). The `__FlashStringHelper` overloads are accepted for Arduino source-compatibility but the pointer is treated as a plain RAM pointer — there is no Harvard-architecture flash on ESP32.

Note

Call begin once before passing this logger to CryptnoxWallet; calling any `print` / `println` method before begin is a no-op.

Warning

In production firmware, replace this logger with a null implementation (a CW_Logger that silently discards all output). Leaving ESP32Logger active on a production device exposes full APDU traces and PIN values on the serial console (LOW-03).

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 48 of file ESP32Logger.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ~ESP32Logger()

```
ESP32Logger::~ESP32Logger ()  [inline], [override]
```
Default destructor.
Definition at line 191 of file ESP32Logger.h.

4.2.3 Member Function Documentation

4.2.3.1 begin()

```
bool ESP32Logger::begin (
            unsigned long baudRate = 115200UL)  [override]
```
Initialise UART0 at the given baud rate.

Parameters

in	*baudRate*	UART baud rate (default `115200`).

Returns

`true` on success, `false` if UART driver installation fails.

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 95 of file ESP32Logger.cpp.
References m_initialized, UART_LOG_PORT, and UART_RX_FLOW_THRESH_NONE.
Referenced by app_main().

4.2.3.2 print() [1/7]

```
void ESP32Logger::print (
            char c)  [override]
```
Print a single character.

Parameters

in	*c*	Character to print.

Definition at line 126 of file ESP32Logger.cpp.
References m_initialized.

4.2.3.3 print() [2/7]

```
void ESP32Logger::print (
            const __FlashStringHelper * str)  [override]
```
Print a PROGMEM string.
On ESP32 the pointer is treated as a plain RAM pointer; no special flash-read logic is applied.

Parameters

in	*str*	NUL-terminated string to print (must not be `NULL`).

Definition at line 112 of file ESP32Logger.cpp.

References m_initialized, and uart_write_str().
Referenced by println(), println(), println(), println(), println(), println(), and println().

4.2.3.4 print() [3/7]

```
void ESP32Logger::print (
            const char * str)  [override]
```

Print a NUL-terminated C string.

Parameters

in	*str*	String to print (must not be NULL).

Definition at line 119 of file ESP32Logger.cpp.
References m_initialized, and uart_write_str().

4.2.3.5 print() [4/7]

```
void ESP32Logger::print (
            int value,
            int base = DEC)  [override]
```

Print a signed integer.
Outputs a '-' prefix for negative values when base is DEC.

Parameters

in	*value*	Integer to print.
in	*base*	Numeric base (see print(uint8_t,int)).

Definition at line 157 of file ESP32Logger.cpp.
References clamp_base(), m_initialized, and write_uint_to_uart().

4.2.3.6 print() [5/7]

```
void ESP32Logger::print (
            uint16_t value,
            int base = DEC)  [override]
```

Print a 16-bit unsigned integer.

Parameters

in	*value*	Integer to print.
in	*base*	Numeric base (see print(uint8_t,int)).

Definition at line 141 of file ESP32Logger.cpp.
References clamp_base(), m_initialized, and write_uint_to_uart().

4.2.3.7 print() [6/7]

```
void ESP32Logger::print (
            uint32_t value,
            int base = DEC)  [override]
```

Print a 32-bit unsigned integer.

Parameters

in	*value*	Integer to print.
in	*base*	Numeric base (see print(uint8_t,int)).

Definition at line 149 of file ESP32Logger.cpp.
References clamp_base(), m_initialized, and write_uint_to_uart().

4.2.3.8 print() [7/7]

```
void ESP32Logger::print (
            uint8_t value,
            int base = DEC)  [override]
```

Print an 8-bit unsigned integer.

Parameters

in	*value*	Integer to print.
in	*base*	Numeric base: DEC (10), HEX (16), OCT (8), or BIN (2). Defaults to DEC.

Definition at line 133 of file ESP32Logger.cpp.
References clamp_base(), m_initialized, and write_uint_to_uart().

4.2.3.9 println() [1/8]

```
void ESP32Logger::println ()  [override]
```

Print a CR+LF newline sequence.
Definition at line 173 of file ESP32Logger.cpp.
References LOGGER_NEWLINE, m_initialized, and uart_write_str().
Referenced by println(), println(), println(), println(), println(), println(), and println().

4.2.3.10 println() [2/8]

```
void ESP32Logger::println (
            char c)  [override]
```

Print a single character followed by a newline.

Parameters

in	*c*	Character to print.

Definition at line 192 of file ESP32Logger.cpp.
References print(), and println().

4.2.3.11 println() [3/8]

```
void ESP32Logger::println (
            const __FlashStringHelper * str)  [override]
```

Print a PROGMEM string followed by a newline.

Parameters

in	*str*	NUL-terminated string to print (must not be NULL).

Definition at line 180 of file ESP32Logger.cpp.

References print(), and println().

4.2.3.12 println() [4/8]

```
void ESP32Logger::println (
            const char * str)  [override]
```

Print a NUL-terminated C string followed by a newline.

Parameters

in	*str*	String to print (must not be NULL).

Definition at line 186 of file ESP32Logger.cpp.
References print(), and println().

4.2.3.13 println() [5/8]

```
void ESP32Logger::println (
            int value,
            int base = DEC)  [override]
```

Print a signed integer followed by a newline.

Parameters

in	*value*	Integer to print.
in	*base*	Numeric base (see print(uint8_t,int)).

Definition at line 216 of file ESP32Logger.cpp.
References print(), and println().

4.2.3.14 println() [6/8]

```
void ESP32Logger::println (
            uint16_t value,
            int base = DEC)  [override]
```

Print a 16-bit unsigned integer followed by a newline.

Parameters

in	*value*	Integer to print.
in	*base*	Numeric base (see print(uint8_t,int)).

Definition at line 204 of file ESP32Logger.cpp.
References print(), and println().

4.2.3.15 println() [7/8]

```
void ESP32Logger::println (
            uint32_t value,
            int base = DEC)  [override]
```

Print a 32-bit unsigned integer followed by a newline.

Parameters

in	*value*	Integer to print.

in	*base*	Numeric base (see print(uint8_t,int)).

Definition at line 210 of file ESP32Logger.cpp.
References print(), and println().

4.2.3.16 println() [8/8]

```
void ESP32Logger::println (
            uint8_t value,
            int base = DEC)  [override]
```

Print an 8-bit unsigned integer followed by a newline.

Parameters

in	*value*	Integer to print.
in	*base*	Numeric base (see print(uint8_t,int)).

Definition at line 198 of file ESP32Logger.cpp.
References print(), and println().

4.2.4 Member Data Documentation

4.2.4.1 m_initialized

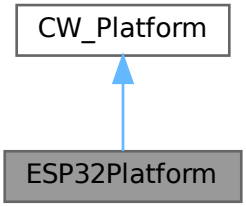
```
bool ESP32Logger::m_initialized = false  [private]
```

`true` after a successful begin call.
Definition at line 194 of file ESP32Logger.h.
Referenced by begin(), print(), print(), print(), print(), print(), print(), print(), and println().
The documentation for this class was generated from the following files:

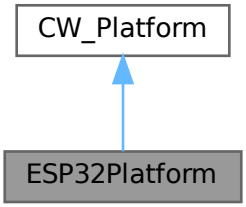
- components/esp32_logger/include/ESP32Logger.h
- components/esp32_logger/ESP32Logger.cpp

4.3 ESP32Platform Class Reference

CW_Platform backed by FreeRTOS `vTaskDelay`.
`#include <ESP32Platform.h>`
Inheritance diagram for ESP32Platform:



Collaboration diagram for ESP32Platform:



Public Member Functions

- void sleep_ms (uint32_t ms) override

  Sleep for at least `ms` milliseconds.
- ~ESP32Platform () override

  Default destructor.

4.3.1 Detailed Description

CW_Platform backed by FreeRTOS `vTaskDelay`.
The sleep_ms implementation delegates to `vTaskDelay(pdMS_TO_TICKS(ms))`, yielding the calling FreeRTOS task to the scheduler for at least the requested duration. The actual sleep may be slightly longer than `ms` due to tick granularity.

Note

Instantiate once and inject into CryptnoxWallet alongside the other adapter objects.

See also

CW_Platform

CryptnoxWallet

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 38 of file ESP32Platform.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ~ESP32Platform()

```
ESP32Platform::~ESP32Platform () [inline], [override]
```

Default destructor.
Definition at line 52 of file ESP32Platform.h.

4.3.3 Member Function Documentation

4.3.3.1 sleep_ms()

```
void ESP32Platform::sleep_ms (
            uint32_t ms)  [override]
```

Sleep for at least `ms` milliseconds.
Yield to the FreeRTOS scheduler for at least `ms` milliseconds.
Calls `vTaskDelay(pdMS_TO_TICKS(ms))`. The calling FreeRTOS task is blocked and yields the CPU to other ready tasks for the duration.

Parameters

`in`	*ms*	Minimum sleep duration in milliseconds. A value of `0` yields once to the scheduler without a guaranteed delay.

Definition at line 20 of file ESP32Platform.cpp.
The documentation for this class was generated from the following files:

- components/esp32_platform/include/ESP32Platform.h
- components/esp32_platform/ESP32Platform.cpp

4.4 eth_tx_t Struct Reference

```
#include <eth_rlp.h>
```

Public Attributes

- uint64_t chain_id
- uint64_t nonce
- uint64_t max_priority_fee
- uint64_t max_fee
- uint64_t gas_limit
- uint8_t to [20]
- uint64_t eth_value
- const uint8_t * calldata
- size_t calldata_len

4.4.1 Detailed Description

Examples

UsdcSigning/main/main.cpp.

Definition at line 16 of file eth_rlp.h.

4.4.2 Member Data Documentation

4.4.2.1 calldata

```
const uint8_t* eth_tx_t::calldata
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 24 of file eth_rlp.h.
Referenced by encode_common_fields(), and signing_loop().

4.4.2.2 calldata_len

```
size_t eth_tx_t::calldata_len
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 25 of file eth_rlp.h.
Referenced by encode_common_fields(), and signing_loop().

4.4.2.3 chain_id

```
uint64_t eth_tx_t::chain_id
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 17 of file eth_rlp.h.
Referenced by encode_common_fields(), and signing_loop().

4.4.2.4 eth_value

```
uint64_t eth_tx_t::eth_value
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 23 of file eth_rlp.h.
Referenced by encode_common_fields(), and signing_loop().

4.4.2.5 gas_limit

```
uint64_t eth_tx_t::gas_limit
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 21 of file eth_rlp.h.
Referenced by encode_common_fields(), and signing_loop().

4.4.2.6 max_fee

```
uint64_t eth_tx_t::max_fee
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 20 of file eth_rlp.h.
Referenced by encode_common_fields(), and signing_loop().

4.4.2.7 max_priority_fee

```
uint64_t eth_tx_t::max_priority_fee
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 19 of file eth_rlp.h.
Referenced by encode_common_fields(), and signing_loop().

4.4.2.8 nonce

```
uint64_t eth_tx_t::nonce
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 18 of file eth_rlp.h.
Referenced by encode_common_fields(), and signing_loop().

4.4.2.9 to

```
uint8_t eth_tx_t::to[20]
```

Examples

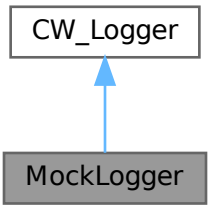
UsdcSigning/main/main.cpp.

Definition at line 22 of file eth_rlp.h.
Referenced by encode_common_fields(), and signing_loop().
The documentation for this struct was generated from the following file:

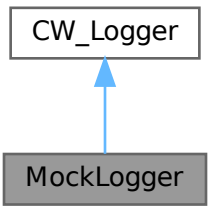
- examples/UsdcSigning/main/eth_rlp.h

4.5 MockLogger Class Reference

Inheritance diagram for MockLogger:



Collaboration diagram for MockLogger:



Public Member Functions

- bool begin (unsigned long=MOCK_UART_BAUD_RATE) override
- void print (const __FlashStringHelper *) override
- void print (const char *) override
- void print (char) override
- void print (uint8_t, int=DEC) override
- void print (uint16_t, int=DEC) override
- void print (uint32_t, int=DEC) override
- void print (int, int=DEC) override
- void println () override
- void println (const __FlashStringHelper *) override
- void println (const char *) override
- void println (char) override
- void println (uint8_t, int=DEC) override
- void println (uint16_t, int=DEC) override
- void println (uint32_t, int=DEC) override
- void println (int, int=DEC) override
- ~MockLogger () override

4.5.1 Detailed Description

Definition at line 96 of file test_cw_secure_channel.cpp.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 ~MockLogger()

```
MockLogger::~MockLogger () [inline], [override]
```

Definition at line 131 of file test_cw_secure_channel.cpp.

4.5.3 Member Function Documentation

4.5.3.1 begin()

```
bool MockLogger::begin (
            unsigned long = MOCK_UART_BAUD_RATE) [inline], [override]
```

Definition at line 98 of file test_cw_secure_channel.cpp.
References MOCK_UART_BAUD_RATE.

4.5.3.2 print() [1/7]

```
void MockLogger::print (
            char ) [inline], [override]
```

Definition at line 105 of file test_cw_secure_channel.cpp.

4.5.3.3 print() [2/7]

```
void MockLogger::print (
            const __FlashStringHelper * ) [inline], [override]
```

Definition at line 101 of file test_cw_secure_channel.cpp.

4.5.3.4 print() [3/7]

```
void MockLogger::print (
            const char * ) [inline], [override]
```

Definition at line 103 of file test_cw_secure_channel.cpp.

4.5.3.5 print() [4/7]

```
void MockLogger::print (
            int ,
            int  = DEC)  [inline], [override]
```

Definition at line 113 of file test_cw_secure_channel.cpp.

4.5.3.6 print() [5/7]

```
void MockLogger::print (
            uint16_t ,
            int  = DEC)  [inline], [override]
```

Definition at line 109 of file test_cw_secure_channel.cpp.

4.5.3.7 print() [6/7]

```
void MockLogger::print (
            uint32_t ,
            int  = DEC)  [inline], [override]
```

Definition at line 111 of file test_cw_secure_channel.cpp.

4.5.3.8 print() [7/7]

```
void MockLogger::print (
            uint8_t ,
            int  = DEC)  [inline], [override]
```

Definition at line 107 of file test_cw_secure_channel.cpp.

4.5.3.9 println() [1/8]

```
void MockLogger::println ()  [inline], [override]
```

Definition at line 115 of file test_cw_secure_channel.cpp.

4.5.3.10 println() [2/8]

```
void MockLogger::println (
            char )  [inline], [override]
```

Definition at line 121 of file test_cw_secure_channel.cpp.

4.5.3.11 println() [3/8]

```
void MockLogger::println (
            const __FlashStringHelper * )  [inline], [override]
```

Definition at line 117 of file test_cw_secure_channel.cpp.

4.5.3.12 println() [4/8]

```
void MockLogger::println (
            const char * )  [inline], [override]
```

Definition at line 119 of file test_cw_secure_channel.cpp.

4.5.3.13 println() [5/8]

```
void MockLogger::println (
            int ,
            int  = DEC)  [inline], [override]
```

Definition at line 129 of file test_cw_secure_channel.cpp.

4.5.3.14 println() [6/8]

```
void MockLogger::println (
            uint16_t ,
            int   = DEC)  [inline], [override]
```

Definition at line 125 of file test_cw_secure_channel.cpp.

4.5.3.15 println() [7/8]

```
void MockLogger::println (
            uint32_t ,
            int   = DEC)  [inline], [override]
```

Definition at line 127 of file test_cw_secure_channel.cpp.

4.5.3.16 println() [8/8]

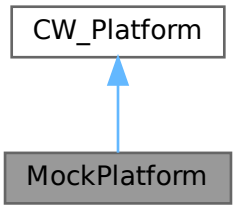
```
void MockLogger::println (
            uint8_t ,
            int   = DEC)  [inline], [override]
```

Definition at line 123 of file test_cw_secure_channel.cpp.
The documentation for this class was generated from the following file:

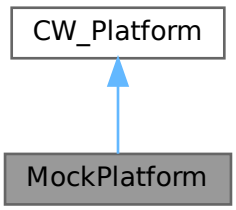
- components/secure_channel/test/test_cw_secure_channel.cpp

4.6 MockPlatform Class Reference

Inheritance diagram for MockPlatform:



Collaboration diagram for MockPlatform:



Public Member Functions

- void sleep_ms (uint32_t) override
- ~MockPlatform () override

4.6.1 Detailed Description

Definition at line 139 of file test_cw_secure_channel.cpp.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 ~MockPlatform()

```
MockPlatform::~MockPlatform ()  [inline], [override]
```

Definition at line 143 of file test_cw_secure_channel.cpp.

4.6.3 Member Function Documentation

4.6.3.1 sleep_ms()

```
void MockPlatform::sleep_ms (
            uint32_t )  [inline], [override]
```

Definition at line 141 of file test_cw_secure_channel.cpp.
The documentation for this class was generated from the following file:

- components/secure_channel/test/test_cw_secure_channel.cpp

4.7 MockScriptEntry Struct Reference

Public Attributes

- uint8_t data [MOCK_MAX_RESP_BYTES]
- uint8_t len
- bool succeed

4.7.1 Detailed Description

Definition at line 151 of file test_cw_secure_channel.cpp.

4.7.2 Member Data Documentation

4.7.2.1 data

```
uint8_t MockScriptEntry::data[MOCK_MAX_RESP_BYTES]
```

Definition at line 152 of file test_cw_secure_channel.cpp.
Referenced by ScriptedMockNfcTransport::sendAPDU().

4.7.2.2 len

```
uint8_t MockScriptEntry::len
```

Definition at line 153 of file test_cw_secure_channel.cpp.
Referenced by ScriptedMockNfcTransport::sendAPDU().

4.7.2.3 succeed

`bool MockScriptEntry::succeed`
Definition at line 154 of file test_cw_secure_channel.cpp.
Referenced by ScriptedMockNfcTransport::sendAPDU().
The documentation for this struct was generated from the following file:

- components/secure_channel/test/test_cw_secure_channel.cpp

4.8 pn532_config_t Struct Reference

Compile-time configuration passed to pn532_init.
`#include <pn532.h>`

Public Attributes

- pn532_transport_t transport
- spi_host_device_t spi_host
- int pin_mosi
- int pin_miso
- int pin_sclk
- int pin_cs
- bool skip_bus_init
- int i2c_port
- int pin_sda
- int pin_scl
- int pin_irq
- int pin_rst
- uint32_t i2c_clock_hz

4.8.1 Detailed Description

Compile-time configuration passed to pn532_init.
Only the fields that correspond to the selected transport need to be initialised; fields for the other transport are ignored.

SPI example

```
pn532_config_t cfg = {};
cfg.transport     = PN532_TRANSPORT_SPI;
cfg.spi_host      = SPI2_HOST;
cfg.pin_cs        = 10;
cfg.skip_bus_init = true;
```

I²C example

```
pn532_config_t cfg = {};
cfg.transport    = PN532_TRANSPORT_I2C;
cfg.i2c_port     = 0;
cfg.pin_sda      = 27;
cfg.pin_scl      = 22;
cfg.pin_irq      = -1;
cfg.pin_rst      = -1;
cfg.i2c_clock_hz = 100000U;
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 111 of file pn532.h.

4.8.2 Member Data Documentation

4.8.2.1 i2c_clock_hz

`uint32_t pn532_config_t::i2c_clock_hz`
I²C clock frequency in Hz (`100000` standard, `400000` fast).

Examples

BasicUsage/main/main.cpp, and UsdcSigning/main/main.cpp.

Definition at line 128 of file pn532.h.
Referenced by app_main(), and pn532_init_i2c().

4.8.2.2 i2c_port

`int pn532_config_t::i2c_port`
I²C port number (`I2C_NUM_0` or `I2C_NUM_1`).

Examples

BasicUsage/main/main.cpp, and UsdcSigning/main/main.cpp.

Definition at line 123 of file pn532.h.
Referenced by app_main(), and pn532_init_i2c().

4.8.2.3 pin_cs

`int pn532_config_t::pin_cs`
Chip-select GPIO number (software-driven).

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 119 of file pn532.h.
Referenced by app_main(), and pn532_init_spi().

4.8.2.4 pin_irq

`int pn532_config_t::pin_irq`
IRQ input GPIO (-1 = unused; IRQ is not currently used by the driver).

Examples

BasicUsage/main/main.cpp, and UsdcSigning/main/main.cpp.

Definition at line 126 of file pn532.h.
Referenced by app_main(), and pn532_init_i2c().

4.8.2.5 pin_miso

`int pn532_config_t::pin_miso`
MISO GPIO number (ignored when `skip_bus_init` is `true`).
Definition at line 117 of file pn532.h.
Referenced by pn532_init_spi().

4.8.2.6 pin_mosi

`int pn532_config_t::pin_mosi`
MOSI GPIO number (ignored when `skip_bus_init` is `true`).
Definition at line 116 of file pn532.h.
Referenced by pn532_init_spi().

4.8.2.7 pin_rst

`int pn532_config_t::pin_rst`
Reset output GPIO (-1 = unused; a pulse is asserted on init when provided).

Examples

BasicUsage/main/main.cpp, and UsdcSigning/main/main.cpp.

Definition at line 127 of file pn532.h.
Referenced by app_main(), and pn532_init_i2c().

4.8.2.8 pin_scl

`int pn532_config_t::pin_scl`
SCL GPIO number.

Examples

BasicUsage/main/main.cpp, and UsdcSigning/main/main.cpp.

Definition at line 125 of file pn532.h.
Referenced by app_main(), and pn532_init_i2c().

4.8.2.9 pin_sclk

`int pn532_config_t::pin_sclk`
SCLK GPIO number (ignored when `skip_bus_init` is `true`).
Definition at line 118 of file pn532.h.
Referenced by pn532_init_spi().

4.8.2.10 pin_sda

`int pn532_config_t::pin_sda`
SDA GPIO number.

Examples

BasicUsage/main/main.cpp, and UsdcSigning/main/main.cpp.

Definition at line 124 of file pn532.h.
Referenced by app_main(), and pn532_init_i2c().

4.8.2.11 skip_bus_init

`bool pn532_config_t::skip_bus_init`
When `true`, skip `spi_bus_initialize()`; caller has already done it.

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 120 of file pn532.h.
Referenced by app_main(), and pn532_init_spi().

4.8.2.12 spi_host

`spi_host_device_t pn532_config_t::spi_host`
SPI peripheral (`SPI2_HOST` or `SPI3_HOST`).

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 115 of file pn532.h.
Referenced by app_main(), and pn532_init_spi().

4.8.2.13 transport

`pn532_transport_t pn532_config_t::transport`
Active transport (PN532_TRANSPORT_SPI or PN532_TRANSPORT_I2C).

Examples

BasicUsage/main/main.cpp, and UsdcSigning/main/main.cpp.

Definition at line 112 of file pn532.h.
Referenced by app_main(), and pn532_init().
The documentation for this struct was generated from the following file:

- components/pn532/include/pn532.h

4.9 pn532_t Struct Reference

Opaque-like runtime state for a single PN532 instance.
`#include <pn532.h>`

Public Attributes

- pn532_transport_t transport
- spi_device_handle_t spi
- int pin_cs
- i2c_master_bus_handle_t i2c_bus
- i2c_master_dev_handle_t i2c_dev
- int pin_irq
- int pin_rst

4.9.1 Detailed Description

Opaque-like runtime state for a single PN532 instance.
Populated by pn532_init; must remain valid for the lifetime of all subsequent driver calls. Do not read or write fields directly — treat this as an opaque handle.

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 141 of file pn532.h.

4.9.2 Member Data Documentation

4.9.2.1 i2c_bus

`i2c_master_bus_handle_t pn532_t::i2c_bus`
I²C bus handle returned by `i2c_new_master_bus()`.
Definition at line 149 of file pn532.h.
Referenced by pn532_init_i2c().

4.9.2.2 i2c_dev

`i2c_master_dev_handle_t pn532_t::i2c_dev`
I²C device handle returned by `i2c_master_bus_add_device()`.
Definition at line 150 of file pn532.h.
Referenced by i2c_read_ready(), pn532_init_i2c(), read_data(), read_data_apdu_frame(), and write_command().

4.9.2.3 pin_cs

`int pn532_t::pin_cs`
Chip-select GPIO (cached from config for fast toggling).
Definition at line 146 of file pn532.h.
Referenced by pn532_init_spi(), read_data(), read_data_apdu_frame(), read_ready(), and write_command().

4.9.2.4 pin_irq

`int pn532_t::pin_irq`
IRQ GPIO (-1 when unused).
Definition at line 151 of file pn532.h.
Referenced by pn532_init_i2c().

4.9.2.5 pin_rst

`int pn532_t::pin_rst`
Reset GPIO (-1 when unused).
Definition at line 152 of file pn532.h.
Referenced by pn532_init_i2c().

4.9.2.6 spi

`spi_device_handle_t pn532_t::spi`
SPI device handle returned by `spi_bus_add_device()`.
Definition at line 145 of file pn532.h.
Referenced by pn532_init_spi(), spi_read_byte(), and spi_write_byte().

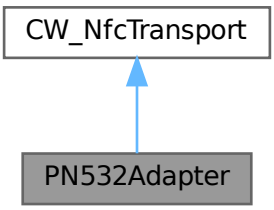
4.9.2.7 transport

`pn532_transport_t pn532_t::transport`
Active transport (copied from pn532_config_t).
Definition at line 142 of file pn532.h.
Referenced by pn532_init(), read_data(), read_data_apdu_frame(), read_ready(), and write_command().
The documentation for this struct was generated from the following file:

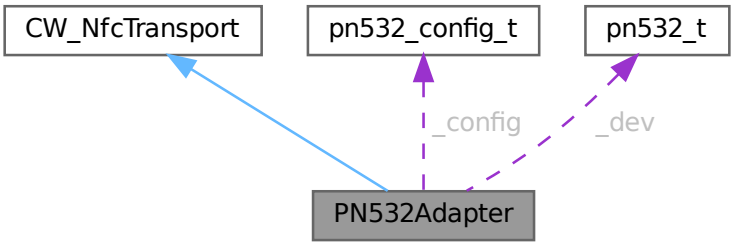
- components/pn532/include/pn532.h

4.10 PN532Adapter Class Reference

Self-contained CW_NfcTransport that owns its pn532_t instance.
`#include <pn532_adapter.h>`
Inheritance diagram for PN532Adapter:



Collaboration diagram for PN532Adapter:



Public Member Functions

- PN532Adapter (const pn532_config_t &config, CW_Logger &logger)

  Construct the adapter with the given PN532 configuration.
- bool begin () override

  Initialise the PN532 driver and configure the SAM.
- bool inListPassiveTarget () override

  Scan for a passive ISO 14443-A card.
- bool sendAPDU (const uint8_t ∗apdu, uint8_t apduLen, uint8_t ∗response, uint8_t &response↩ Len) override

  Exchange one ISO-DEP APDU with the selected card (short response).
- bool sendAPDULarge (const uint8_t ∗apdu, uint8_t apduLen, uint8_t ∗response, uint16_↩ t &responseLen) override

  Exchange one ISO-DEP APDU with the selected card (large response).
- void resetReader () override

  Release the currently selected NFC target.
- bool printFirmwareVersion () override

  Query and log the PN532 firmware version.

Private Attributes

- pn532_config_t _config

  Stored configuration forwarded to pn532_init.
- pn532_t _dev

  PN532 device state (owned by this adapter).
- CW_Logger & _logger

  Logger reference for printFirmwareVersion.
- bool _initialized

  `true` *after a successful pn532_init call.*

4.10.1 Detailed Description

Self-contained CW_NfcTransport that owns its pn532_t instance.
Unlike Pn532NfcTransport (which takes a pre-initialised pn532_t pointer), PN532Adapter accepts a pn532_config_t at construction time and calls pn532_init internally on the first begin() call. This is convenient when the NFC reader is the only SPI/I^{2}C device and the application does not need to manage the bus lifetime separately.

Example

```
pn532_config_t cfg = {};
cfg.transport = PN532_TRANSPORT_SPI;
cfg.spi_host  = SPI2_HOST;
cfg.pin_cs    = 10;

ESP32Logger     logger;
PN532Adapter    transport(cfg, logger);
CryptnoxWallet  wallet(transport, logger, crypto, platform);
```

Note

Non-copyable: `_dev` holds raw SPI/I²C handles that cannot be duplicated.

See also

Pn532NfcTransport

CW_NfcTransport

Definition at line 54 of file pn532_adapter.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 PN532Adapter()

```
PN532Adapter::PN532Adapter (
            const pn532_config_t & config,
            CW_Logger & logger)
```

Construct the adapter with the given PN532 configuration.
Stores the configuration and logger for use during begin(). The PN532 driver is not initialised at this point.

Parameters

in	*config*	Transport and pin configuration forwarded to pn532_init on the first begin() call.
in	*logger*	Logger used by printFirmwareVersion.

Definition at line 27 of file pn532_adapter.cpp.
References _config, _dev, _initialized, and _logger.

4.10.3 Member Function Documentation

4.10.3.1 begin()

```
bool PN532Adapter::begin ()  [override]
```

Initialise the PN532 driver and configure the SAM.
Calls pn532_init (if not already done) followed by pn532_sam_config. Idempotent — repeated calls after a successful init are no-ops that return `true`.

Returns

`true` on success, `false` if pn532_init or pn532_sam_config fails.

Definition at line 32 of file pn532_adapter.cpp.
References _config, _dev, _initialized, pn532_init(), and TAG.

4.10.3.2 inListPassiveTarget()

```
bool PN532Adapter::inListPassiveTarget ()  [override]
```

Scan for a passive ISO 14443-A card.

Returns

`true` if a card was detected in the RF field, `false` if no card is present or a communication error occurred.

Definition at line 48 of file pn532_adapter.cpp.
References _dev, _initialized, PN532_MIFARE_ISO14443A, and pn532_read_passive_target_id().

4.10.3.3 printFirmwareVersion()

```
bool PN532Adapter::printFirmwareVersion () [override]
```

Query and log the PN532 firmware version.
Prints `"PN5xx firmware vX.Y"` via the injected logger.

Returns

`true` if the PN532 returned a valid version, `false` on communication failure or if the adapter has not been initialised.

Definition at line 95 of file pn532_adapter.cpp.
References _dev, _initialized, _logger, FW_BYTE_MASK, FW_IC_SHIFT, FW_REV_SHIFT, FW_VER_SHIFT, and pn532_get_firmware_version().

4.10.3.4 resetReader()

```
void PN532Adapter::resetReader () [override]
```

Release the currently selected NFC target.
Calls pn532_release_target to drop the logical link so the PN532 can list a new target on the next inListPassiveTarget call.
Definition at line 88 of file pn532_adapter.cpp.
References _dev, _initialized, and pn532_release_target().

4.10.3.5 sendAPDU()

```
bool PN532Adapter::sendAPDU (
            const uint8_t * apdu,
            uint8_t apduLen,
            uint8_t * response,
            uint8_t & responseLen) [override]
```

Exchange one ISO-DEP APDU with the selected card (short response).

Parameters

`in`	*apdu*	APDU command bytes (must not be `NULL`).
`in`	*apduLen*	Length of `apdu` ($\leq$ PN532_MAX_APDU_LEN).
`out`	*response*	Caller-allocated buffer for the DataOut bytes.
`in,out`	*responseLen*	In: capacity of `response`. Out: number of DataOut bytes written, capped at `UINT8_MAX` on overflow.

Returns

`true` on success, `false` on transport or card error.

See also

sendAPDULarge

Definition at line 61 of file pn532_adapter.cpp.
References _dev, _initialized, and pn532_send_apdu().

4.10.3.6 sendAPDULarge()

```
bool PN532Adapter::sendAPDULarge (
            const uint8_t * apdu,
            uint8_t apduLen,
            uint8_t * response,
            uint16_t & responseLen)  [override]
```

Exchange one ISO-DEP APDU with the selected card (large response).
Use when the card's DataOut may exceed 255 bytes (e.g. `GET_MANUFACTURER_CERTIFICATE`).

Parameters

`in`	*apdu*	APDU command bytes (must not be `NULL`).
`in`	*apduLen*	Length of `apdu` ($\leq$ PN532_MAX_APDU_LEN).
`out`	*response*	Caller-allocated buffer for the DataOut bytes.
`in,out`	*responseLen*	In: capacity of `response` in bytes. Out: number of DataOut bytes actually written.

Returns

`true` on success, `false` on transport or card error.

See also

sendAPDU

Definition at line 76 of file pn532_adapter.cpp.
References _dev, _initialized, and pn532_send_apdu().

4.10.4 Member Data Documentation

4.10.4.1 _config

```
pn532_config_t PN532Adapter::_config  [private]
```

Stored configuration forwarded to pn532_init.
Definition at line 141 of file pn532_adapter.h.
Referenced by begin(), and PN532Adapter().

4.10.4.2 _dev

```
pn532_t PN532Adapter::_dev  [private]
```

PN532 device state (owned by this adapter).
Definition at line 142 of file pn532_adapter.h.
Referenced by begin(), inListPassiveTarget(), PN532Adapter(), printFirmwareVersion(), resetReader(), sendAPDU(), and sendAPDULarge().

4.10.4.3 _initialized

```
bool PN532Adapter::_initialized  [private]
```

`true` after a successful pn532_init call.
Definition at line 144 of file pn532_adapter.h.
Referenced by begin(), inListPassiveTarget(), PN532Adapter(), printFirmwareVersion(), resetReader(), sendAPDU(), and sendAPDULarge().

4.10.4.4 _logger

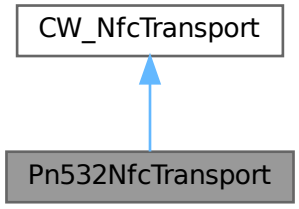
```
CW_Logger& PN532Adapter::_logger  [private]
```

Logger reference for printFirmwareVersion.
Definition at line 143 of file pn532_adapter.h.
Referenced by PN532Adapter(), and printFirmwareVersion().
The documentation for this class was generated from the following files:

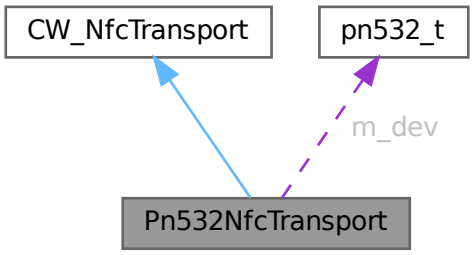
- components/pn532_adapter/include/pn532_adapter.h
- components/pn532_adapter/pn532_adapter.cpp

4.11 Pn532NfcTransport Class Reference

CW_NfcTransport implementation backed by the ESP-IDF PN532 driver.
`#include <Pn532NfcTransport.h>`
Inheritance diagram for Pn532NfcTransport:



Collaboration diagram for Pn532NfcTransport:



Public Member Functions

- Pn532NfcTransport (pn532_t *dev, CW_Logger &logger)
  Construct the transport adapter.
- bool begin () override
  Configure the PN532 SAM and prepare it to accept card commands.
- bool inListPassiveTarget () override
  Scan for a passive ISO 14443-A card.
- bool sendAPDU (const uint8_t *apdu, uint8_t apduLen, uint8_t *response, uint8_t &responseLen) override

Exchange one ISO-DEP APDU with the selected card (short response).

- bool sendAPDULarge (const uint8_t *apdu, uint8_t apduLen, uint8_t *response, uint16_t &responseLen) override

  Exchange one ISO-DEP APDU with the selected card (large response).
- void resetReader () override

  Release the currently selected NFC target.
- bool printFirmwareVersion () override

  Query and log the PN532 firmware version.
- ~Pn532NfcTransport () override

  Default destructor.

Private Attributes

- pn532_t * m_dev

  PN532 device handle (not owned; must outlive this object).
- CW_Logger & m_logger

  Logger for printFirmwareVersion output.

4.11.1 Detailed Description

CW_NfcTransport implementation backed by the ESP-IDF PN532 driver.
Delegates every NFC operation to the C-level PN532 NFC driver via a pn532_t handle supplied by the caller. The handle's lifetime must exceed the lifetime of this object.
Both sendAPDU (response $\leq$ 255 bytes, `uint8_t` length) and sendAPDULarge (response $\leq$ 65535 bytes, `uint16_t` length) are implemented; both ultimately call pn532_send_apdu — the difference is only in the type used to report the response length back to the caller.

Note

Non-copyable: the class holds a reference (`CW_Logger &`) and a raw pointer (`pn532_t *`) that cannot be transferred safely.

See also

pn532_t

CW_NfcTransport

CryptnoxWallet

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 80 of file Pn532NfcTransport.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Pn532NfcTransport()

```
Pn532NfcTransport::Pn532NfcTransport (
            pn532_t * dev,
            CW_Logger & logger)
```

Construct the transport adapter.

Parameters

in	*dev*	Pointer to an already-initialised pn532_t handle. Must remain valid for the lifetime of this object.
in	*logger*	Logger used by printFirmwareVersion.

Definition at line 22 of file Pn532NfcTransport.cpp.
References m_dev, and m_logger.

4.11.2.2 ~Pn532NfcTransport()

`Pn532NfcTransport::~Pn532NfcTransport () [inline], [override]`
Default destructor.
Definition at line 173 of file Pn532NfcTransport.h.

4.11.3 Member Function Documentation

4.11.3.1 begin()

`bool Pn532NfcTransport::begin () [override]`
Configure the PN532 SAM and prepare it to accept card commands.
Calls pn532_sam_config. Must succeed before inListPassiveTarget or any APDU exchange is attempted. Called automatically by `CryptnoxWallet::begin()`.

Returns

`true` if the PN532 acknowledged SAMConfiguration, `false` otherwise.

Definition at line 27 of file Pn532NfcTransport.cpp.
References m_dev, and pn532_sam_config().

4.11.3.2 inListPassiveTarget()

`bool Pn532NfcTransport::inListPassiveTarget () [override]`
Scan for a passive ISO 14443-A card.
Calls pn532_read_passive_target_id with the ISO 14443-A baud-rate selector.

Returns

`true` if at least one card was found in the RF field, `false` if no card is present or a communication error occurred.

Definition at line 33 of file Pn532NfcTransport.cpp.
References m_dev, PN532_MIFARE_ISO14443A, and pn532_read_passive_target_id().

4.11.3.3 printFirmwareVersion()

`bool Pn532NfcTransport::printFirmwareVersion () [override]`
Query and log the PN532 firmware version.
Calls pn532_get_firmware_version and prints a human-readable version string (`"PN5xx firmware vX.Y"`) via the injected logger.

Returns

`true` if the PN532 returned a non-zero version, `false` on communication failure.

Definition at line 61 of file Pn532NfcTransport.cpp.
References m_dev, m_logger, PN532_BYTE_MASK, PN532_FW_IC_SHIFT, PN532_FW_REV_SHIFT, PN532_FW_VER_SHIFT, and pn532_get_firmware_version().
Referenced by app_main().

4.11.3.4 resetReader()

```
void Pn532NfcTransport::resetReader ()  [override]
```

Release the currently selected NFC target.
Calls pn532_release_target so the PN532 drops its logical link to the card and is ready for the next inListPassiveTarget call.
Definition at line 56 of file Pn532NfcTransport.cpp.
References m_dev, and pn532_release_target().

4.11.3.5 sendAPDU()

```
bool Pn532NfcTransport::sendAPDU (
            const uint8_t * apdu,
            uint8_t apduLen,
            uint8_t * response,
            uint8_t & responseLen)  [override]
```

Exchange one ISO-DEP APDU with the selected card (short response).
Use this overload when the card's DataOut is guaranteed to fit in a `uint8_t` ($\leq$ 255 bytes). For larger responses — such as the manufacturer certificate — use sendAPDULarge.

Parameters

`in`	*apdu*	APDU command bytes (must not be `NULL`).
`in`	*apduLen*	Length of `apdu` in bytes ($\leq$ PN532_MAX_APDU_LEN).
`out`	*response*	Caller-allocated buffer for the DataOut bytes.
`in,out`	*responseLen*	In: capacity of `response`. Out: number of DataOut bytes written, capped at `UINT8_MAX` on overflow.

Returns

`true` on a successful APDU exchange, `false` otherwise.

See also

sendAPDULarge

Definition at line 39 of file Pn532NfcTransport.cpp.
References m_dev, and pn532_send_apdu().

4.11.3.6 sendAPDULarge()

```
bool Pn532NfcTransport::sendAPDULarge (
            const uint8_t * apdu,
            uint8_t apduLen,
            uint8_t * response,
            uint16_t & responseLen)  [override]
```

Exchange one ISO-DEP APDU with the selected card (large response).
Identical to sendAPDU but uses a `uint16_t` response length, allowing DataOut up to 65535 bytes. Used for commands whose response can exceed 255 bytes (e.g. `GET_MANUFACTURER_↩ CERTIFICATE` whose DataOut may be ~415 bytes and requires an extended PN532 frame).

Parameters

`in`	*apdu*	APDU command bytes (must not be `NULL`).
`in`	*apduLen*	Length of `apdu` in bytes ($\leq$ PN532_MAX_APDU_LEN).
`out`	*response*	Caller-allocated buffer for the DataOut bytes.

`in,out`	*responseLen*	In: capacity of `response` in bytes. Out: number of DataOut bytes actually written.

Returns

`true` on a successful APDU exchange, `false` otherwise.

See also

sendAPDU

Definition at line 49 of file Pn532NfcTransport.cpp.
References m_dev, and pn532_send_apdu().

4.11.4 Member Data Documentation

4.11.4.1 m_dev

```
pn532_t* Pn532NfcTransport::m_dev  [private]
```

PN532 device handle (not owned; must outlive this object).
Definition at line 176 of file Pn532NfcTransport.h.
Referenced by begin(), inListPassiveTarget(), Pn532NfcTransport(), printFirmwareVersion(), resetReader(), sendAPDU(), and sendAPDULarge().

4.11.4.2 m_logger

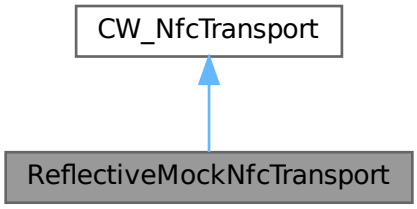
```
CW_Logger& Pn532NfcTransport::m_logger  [private]
```

Logger for printFirmwareVersion output.
Definition at line 177 of file Pn532NfcTransport.h.
Referenced by Pn532NfcTransport(), and printFirmwareVersion().
The documentation for this class was generated from the following files:

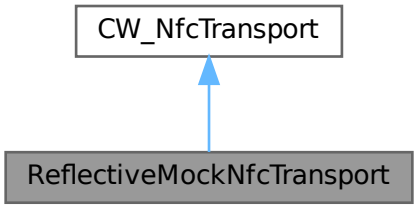
- components/pn532_nfc_transport/include/Pn532NfcTransport.h
- components/pn532_nfc_transport/Pn532NfcTransport.cpp

4.12 ReflectiveMockNfcTransport Class Reference

Inheritance diagram for ReflectiveMockNfcTransport:



Collaboration diagram for ReflectiveMockNfcTransport:



Public Member Functions

- bool begin () override
- bool inListPassiveTarget () override
- void resetReader () override
- bool printFirmwareVersion () override
- bool sendAPDU (const uint8_t *apdu, uint8_t apduLen, uint8_t *response, uint8_t &responseLen) override
- ~ReflectiveMockNfcTransport () override

Public Attributes

- const CW_SecureSession * session = nullptr
- CW_CryptoProvider * crypto = nullptr

4.12.1 Detailed Description

Definition at line 224 of file test_cw_secure_channel.cpp.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 ~ReflectiveMockNfcTransport()

```
ReflectiveMockNfcTransport::~ReflectiveMockNfcTransport () [inline], [override]
```

Definition at line 306 of file test_cw_secure_channel.cpp.

4.12.3 Member Function Documentation

4.12.3.1 begin()

```
bool ReflectiveMockNfcTransport::begin () [inline], [override]
```

Definition at line 229 of file test_cw_secure_channel.cpp.

4.12.3.2 inListPassiveTarget()

```
bool ReflectiveMockNfcTransport::inListPassiveTarget () [inline], [override]
```

Definition at line 232 of file test_cw_secure_channel.cpp.

4.12.3.3 printFirmwareVersion()

```
bool ReflectiveMockNfcTransport::printFirmwareVersion () [inline], [override]
```

Definition at line 237 of file test_cw_secure_channel.cpp.

4.12.3.4 resetReader()

```
void ReflectiveMockNfcTransport::resetReader ()  [inline], [override]
```
Definition at line 235 of file test_cw_secure_channel.cpp.

4.12.3.5 sendAPDU()

```
bool ReflectiveMockNfcTransport::sendAPDU (
            const uint8_t * apdu,
            uint8_t apduLen,
            uint8_t * response,
            uint8_t & responseLen)  [inline], [override]
```
Definition at line 241 of file test_cw_secure_channel.cpp.
References crypto, K_CARD_RESP_PLAINTEXT, SC_AES_BLOCK_BYTES, SC_APDU_MAC_BYTES, SC_APDU_MAC_OFFSET, SC_SW_BYTES, and session.

4.12.4 Member Data Documentation

4.12.4.1 crypto

```
CW_CryptoProvider* ReflectiveMockNfcTransport::crypto = nullptr
```
Definition at line 227 of file test_cw_secure_channel.cpp.
Referenced by sendAPDU().

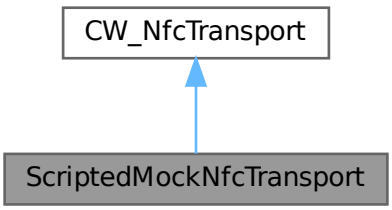
4.12.4.2 session

```
const CW_SecureSession* ReflectiveMockNfcTransport::session = nullptr
```
Definition at line 226 of file test_cw_secure_channel.cpp.
Referenced by sendAPDU().
The documentation for this class was generated from the following file:

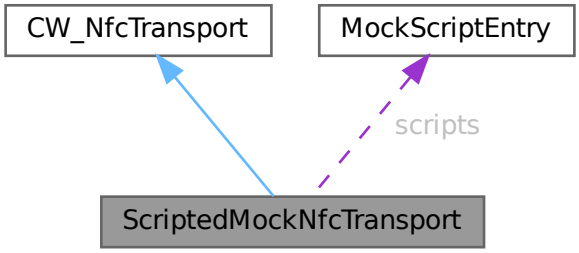
- components/secure_channel/test/test_cw_secure_channel.cpp

4.13 ScriptedMockNfcTransport Class Reference

Inheritance diagram for ScriptedMockNfcTransport:



Collaboration diagram for ScriptedMockNfcTransport:



Public Member Functions

- void reset ()
- void addScript (const uint8_t *data, uint8_t len, bool succeed=true)
- bool begin () override
- bool inListPassiveTarget () override
- void resetReader () override
- bool printFirmwareVersion () override
- bool sendAPDU (const uint8_t *apdu, uint8_t apduLen, uint8_t *response, uint8_t &responseLen) override
- ~ScriptedMockNfcTransport () override

Public Attributes

- MockScriptEntry scripts [MOCK_MAX_SCRIPTS] {}
- uint8_t scriptCount = 0U
- uint8_t callIdx = 0U

4.13.1 Detailed Description

Definition at line 157 of file test_cw_secure_channel.cpp.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 ~ScriptedMockNfcTransport()

```
ScriptedMockNfcTransport::~ScriptedMockNfcTransport () [inline], [override]
```

Definition at line 208 of file test_cw_secure_channel.cpp.

4.13.3 Member Function Documentation

4.13.3.1 addScript()

```
void ScriptedMockNfcTransport::addScript (
            const uint8_t * data,
            uint8_t len,
            bool succeed = true) [inline]
```

Definition at line 169 of file test_cw_secure_channel.cpp.
References MOCK_MAX_SCRIPTS, scriptCount, and scripts.

4.13.3.2 begin()

```
bool ScriptedMockNfcTransport::begin () [inline], [override]
```
Definition at line 178 of file test_cw_secure_channel.cpp.

4.13.3.3 inListPassiveTarget()

```
bool ScriptedMockNfcTransport::inListPassiveTarget () [inline], [override]
```
Definition at line 181 of file test_cw_secure_channel.cpp.

4.13.3.4 printFirmwareVersion()

```
bool ScriptedMockNfcTransport::printFirmwareVersion () [inline], [override]
```
Definition at line 186 of file test_cw_secure_channel.cpp.

4.13.3.5 reset()

```
void ScriptedMockNfcTransport::reset () [inline]
```
Definition at line 163 of file test_cw_secure_channel.cpp.
References callIdx, scriptCount, and scripts.

4.13.3.6 resetReader()

```
void ScriptedMockNfcTransport::resetReader () [inline], [override]
```
Definition at line 184 of file test_cw_secure_channel.cpp.

4.13.3.7 sendAPDU()

```
bool ScriptedMockNfcTransport::sendAPDU (
            const uint8_t * apdu,
            uint8_t apduLen,
            uint8_t * response,
            uint8_t & responseLen) [inline], [override]
```
Definition at line 190 of file test_cw_secure_channel.cpp.
References callIdx, MockScriptEntry::data, MockScriptEntry::len, scriptCount, scripts, and MockScriptEntry::succeed.

4.13.4 Member Data Documentation

4.13.4.1 callIdx

```
uint8_t ScriptedMockNfcTransport::callIdx = 0U
```
Definition at line 161 of file test_cw_secure_channel.cpp.
Referenced by reset(), and sendAPDU().

4.13.4.2 scriptCount

```
uint8_t ScriptedMockNfcTransport::scriptCount = 0U
```
Definition at line 160 of file test_cw_secure_channel.cpp.
Referenced by addScript(), reset(), and sendAPDU().

4.13.4.3 scripts

```
MockScriptEntry ScriptedMockNfcTransport::scripts[MOCK_MAX_SCRIPTS] {}
```
Definition at line 159 of file test_cw_secure_channel.cpp.
Referenced by addScript(), reset(), and sendAPDU().
The documentation for this class was generated from the following file:

- components/secure_channel/test/test_cw_secure_channel.cpp

4.14 uECC_Curve_t Struct Reference

Public Attributes

- mbedtls_ecp_group_id grp_id

4.14.1 Detailed Description

Definition at line 35 of file uECC_esp32.cpp.

4.14.2 Member Data Documentation

4.14.2.1 grp_id

`mbedtls_ecp_group_id uECC_Curve_t::grp_id`
Definition at line 36 of file uECC_esp32.cpp.
Referenced by uECC_make_key(), uECC_shared_secret(), and uECC_verify().
The documentation for this struct was generated from the following file:

- components/uECC/uECC_esp32.cpp

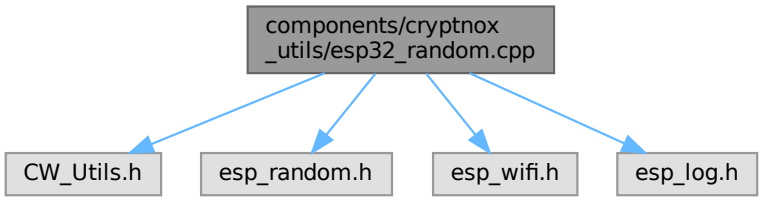
Chapter 5

File Documentation

5.1 components/cryptnox_utils/esp32_random.cpp File Reference

```
#include "CW_Utils.h"
#include "esp_random.h"
#include "esp_wifi.h"
#include "esp_log.h"
```

Include dependency graph for esp32_random.cpp:



Variables

- static const char *const TAG = "CW_Utils"

5.1.1 Variable Documentation

5.1.1.1 TAG

```
const char* const TAG = "CW_Utils"  [static]
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 14 of file esp32_random.cpp.
Referenced by app_main(), PN532Adapter::begin(), do_post(), eth_rpc_ecrecover_parity(), eth_rpc_get_nonce(), eth_rpc_send_raw_tx(), eth_rpc_wifi_connect(), run_basic_usage_loop(), run_connect_loop(), run_sign_loop(), run_verify_pin_loop(), signing_loop(), wifi_event_handler(), and wifi_start().

5.2 esp32_random.cpp

Go to the documentation of this file.

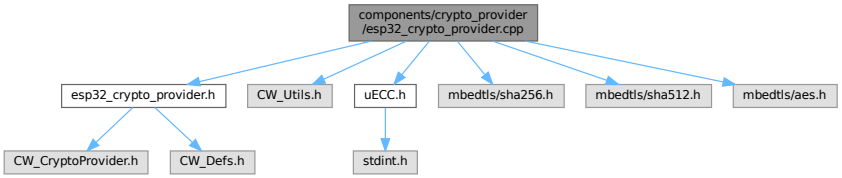
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #include "CW_Utils.h"
00007 #include "esp_random.h"
00008 #include "esp_wifi.h"
00009 #ifdef CONFIG_BT_ENABLED
00010 #  include "esp_bt.h"
00011 #endif
00012 #include "esp_log.h"
00013
00014 static const char* const TAG = "CW_Utils";
00015
00016 bool CW_Utils::fill_secure_random(uint8_t *dest, size_t len) {
00017     bool result = false;
00018
00019     if ((dest != nullptr) && (len != 0U)) {
00020         wifi_mode_t mode    = WIFI_MODE_NULL;
00021         bool        wifi_on = ((esp_wifi_get_mode(&mode) == ESP_OK) && (mode != WIFI_MODE_NULL));
00022
00023 #ifdef CONFIG_BT_ENABLED
00024         bool bt_on = (esp_bt_controller_get_status() == ESP_BT_CONTROLLER_STATUS_ENABLED);
00025 #else
00026         bool bt_on = false;
00027 #endif
00028
00029         /* ESP32-S3 TRNG operates from thermal noise and ring-oscillator jitter
00030          * even without a radio, but a live WiFi/BT subsystem feeds additional
00031          * hardware entropy.  Always fill the buffer; warn when reduced-entropy
00032          * mode is active so callers are aware. */
00033         if ((!wifi_on) && (!bt_on)) {
00034             ESP_LOGW(TAG, "RNG: no radio active -- reduced entropy (TRNG only)");
00035         }
00036
00037         esp_fill_random(dest, len);
00038         result = true;
00039     }
00040
00041     return result;
00042 }
```

5.3 components/crypto_provider/esp32_crypto_provider.cpp File Reference

Implementation of ESP32CryptoProvider — mbedTLS + hardware TRNG backend.

```
#include "esp32_crypto_provider.h"
#include "CW_Utils.h"
#include "uECC.h"
#include "mbedtls/sha256.h"
#include "mbedtls/sha512.h"
#include "mbedtls/aes.h"
```

Include dependency graph for esp32_crypto_provider.cpp:



Macros

- #define AES_BLOCK_SIZE_BYTES (16U) /* bytes in one AES block */
- #define AES_KEY_BITS_PER_BYTE (8U) /* multiplier: key bytes → key bits */
- #define AES_PAD_BUF_MAX_INPUT (256U)
- #define AES_PAD_BUF_SIZE (AES_PAD_BUF_MAX_INPUT + AES_BLOCK_SIZE_BYTES)
- #define BIT_PADDING_MARKER (0x80U) /* mandatory leading 1-bit as a full byte */
- #define PADDING_ZERO_FILL (0x00U) /* fill value for padding bytes after marker */
- #define MBEDTLS_SHA256_MODE (0) /* 0 = SHA-256, 1 = SHA-224 */
- #define MBEDTLS_SHA512_MODE (0) /* 0 = SHA-512, 1 = SHA-384 */
- #define MBEDTLS_OK (0)
- #define UECC_SUCCESS (1)

Functions

- static const uECC_Curve_t * toCurve (CW_Curve curve)

5.3.1 Detailed Description

Implementation of ESP32CryptoProvider — mbedTLS + hardware TRNG backend.
Stitches together the ESP-IDF cryptographic stack behind the single platform-independent CW_↩ CryptoProvider interface. Full API documentation lives on the declarations in esp32_crypto_provider.h.
Definition in file esp32_crypto_provider.cpp.

5.3.2 Macro Definition Documentation

5.3.2.1 AES_BLOCK_SIZE_BYTES

```
#define AES_BLOCK_SIZE_BYTES (16U) /* bytes in one AES block */
```

Definition at line 47 of file esp32_crypto_provider.cpp.
Referenced by ESP32CryptoProvider::aesCbcDecrypt(), and ESP32CryptoProvider::aesCbcEncrypt().

5.3.2.2 AES_KEY_BITS_PER_BYTE

```
#define AES_KEY_BITS_PER_BYTE (8U) /* multiplier:  key bytes → key bits */
```

Definition at line 48 of file esp32_crypto_provider.cpp.
Referenced by ESP32CryptoProvider::aesCbcDecrypt(), and ESP32CryptoProvider::aesCbcEncrypt().

5.3.2.3 AES_PAD_BUF_MAX_INPUT

```
#define AES_PAD_BUF_MAX_INPUT (256U)
```

Definition at line 50 of file esp32_crypto_provider.cpp.

5.3.2.4 AES_PAD_BUF_SIZE

```
#define AES_PAD_BUF_SIZE (AES_PAD_BUF_MAX_INPUT + AES_BLOCK_SIZE_BYTES)
```

Definition at line 52 of file esp32_crypto_provider.cpp.
Referenced by ESP32CryptoProvider::aesCbcEncrypt().

5.3.2.5 BIT_PADDING_MARKER

```
#define BIT_PADDING_MARKER (0x80U) /* mandatory leading 1-bit as a full byte */
```

Definition at line 55 of file esp32_crypto_provider.cpp.
Referenced by ESP32CryptoProvider::aesCbcDecrypt(), and ESP32CryptoProvider::aesCbcEncrypt().

5.3.2.6 MBEDTLS_OK

```
#define MBEDTLS_OK (0)
```
Definition at line 63 of file esp32_crypto_provider.cpp.
Referenced by ESP32CryptoProvider::aesCbcDecrypt(), ESP32CryptoProvider::aesCbcEncrypt(), esp32_mbedtls_rng(), ESP32CryptoProvider::sha256(), ESP32CryptoProvider::sha512(), uECC_make_key(), uECC_shared_secret(), and uECC_verify().

5.3.2.7 MBEDTLS_SHA256_MODE

```
#define MBEDTLS_SHA256_MODE (0) /* 0 = SHA-256, 1 = SHA-224 */
```
Definition at line 59 of file esp32_crypto_provider.cpp.
Referenced by ESP32CryptoProvider::sha256().

5.3.2.8 MBEDTLS_SHA512_MODE

```
#define MBEDTLS_SHA512_MODE (0) /* 0 = SHA-512, 1 = SHA-384 */
```
Definition at line 60 of file esp32_crypto_provider.cpp.
Referenced by ESP32CryptoProvider::sha512().

5.3.2.9 PADDING_ZERO_FILL

```
#define PADDING_ZERO_FILL (0x00U) /* fill value for padding bytes after marker */
```
Definition at line 56 of file esp32_crypto_provider.cpp.
Referenced by ESP32CryptoProvider::aesCbcDecrypt(), and ESP32CryptoProvider::aesCbcEncrypt().

5.3.2.10 UECC_SUCCESS

```
#define UECC_SUCCESS (1)
```
Definition at line 66 of file esp32_crypto_provider.cpp.
Referenced by ESP32CryptoProvider::ecdh(), ESP32CryptoProvider::ecdsaVerify(), ESP32CryptoProvider::makeKey(), uECC_make_key(), uECC_shared_secret(), and uECC_verify().

5.3.3 Function Documentation

5.3.3.1 toCurve()

```
const uECC_Curve_t * toCurve (
            CW_Curve curve)  [static]
```
Definition at line 26 of file esp32_crypto_provider.cpp.
References uECC_secp256k1(), and uECC_secp256r1().
Referenced by ESP32CryptoProvider::ecdh(), ESP32CryptoProvider::ecdsaVerify(), and ESP32CryptoProvider::makeKey().

5.4 esp32_crypto_provider.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00014
00015 #include "esp32_crypto_provider.h"
00016 #include "CW_Utils.h"
00017 #include "uECC.h"
00018 #include "mbedtls/sha256.h"
00019 #include "mbedtls/sha512.h"
00020 #include "mbedtls/aes.h"
00021
00022 /*****************************************************************
00023  * 1b. Internal curve translator
00024  *****************************************************************/
```

```
00025
00026 static const uECC_Curve_t* toCurve(CW_Curve curve) {
00027     const uECC_Curve_t* result = NULL;
00028     switch (curve) {
00029         case CW_CURVE_SECP256R1:
00030             result = uECC_secp256r1();
00031             break;
00032         case CW_CURVE_SECP256K1:
00033             result = uECC_secp256k1();
00034             break;
00035         default:
00036             result = NULL;
00037             break;
00038     }
00039     return result;
00040 }
00041
00042 /*********************************************************************
00043  * 1. Module constants
00044  *********************************************************************/
00045
00046 /* AES */
00047 #define AES_BLOCK_SIZE_BYTES      (16U)  /* bytes in one AES block                          */
00048 #define AES_KEY_BITS_PER_BYTE     (8U)   /* multiplier: key bytes → key bits                */
00049 /* Maximum AES plaintext that can be padded: covers CW_USER_DATA_PAGE_SIZE + one block. */
00050 #define AES_PAD_BUF_MAX_INPUT     (256U)
00051 /* Pad buffer: max input + one extra block for the bit-padding byte.                    */
00052 #define AES_PAD_BUF_SIZE          (AES_PAD_BUF_MAX_INPUT + AES_BLOCK_SIZE_BYTES)
00053
00054 /* ISO/IEC 9797-1 Method 2 (bit padding) */
00055 #define BIT_PADDING_MARKER        (0x80U)  /* mandatory leading 1-bit as a full byte */
00056 #define PADDING_ZERO_FILL         (0x00U)  /* fill value for padding bytes after marker */
00057
00058 /* mbedTLS SHA mode selectors */
00059 #define MBEDTLS_SHA256_MODE       (0)   /* 0 = SHA-256, 1 = SHA-224 */
00060 #define MBEDTLS_SHA512_MODE       (0)   /* 0 = SHA-512, 1 = SHA-384 */
00061
00062 /* mbedTLS return code for success */
00063 #define MBEDTLS_OK                (0)
00064
00065 /* uECC return code for success */
00066 #define UECC_SUCCESS              (1)
00067
00068 /*********************************************************************
00069  * 2. SHA methods
00070  *********************************************************************/
00071
00073 bool ESP32CryptoProvider::sha256(const uint8_t* data, size_t len, uint8_t* out) {
00074     int ret = mbedtls_sha256(data, len, out, MBEDTLS_SHA256_MODE);
00075     return (ret == MBEDTLS_OK);
00076 }
00077
00079 bool ESP32CryptoProvider::sha512(const uint8_t* data, size_t len, uint8_t* out) {
00080     int ret = mbedtls_sha512(data, len, out, MBEDTLS_SHA512_MODE);
00081     return (ret == MBEDTLS_OK);
00082 }
00083
00084 /*********************************************************************
00085  * 3. AES-CBC encrypt
00086  *********************************************************************/
00087
00089 uint16_t ESP32CryptoProvider::aesCbcEncrypt(const uint8_t* in, uint16_t len, uint8_t* out,
00090                                             const uint8_t* key, uint8_t keyLen,
00091                                             uint8_t* iv, bool bitPadding) {
00092     uint16_t result = 0U;
00093
00094     if ((in != NULL) && (out != NULL) && (key != NULL) && (iv != NULL)) {
00095         uint8_t  padBuf[AES_PAD_BUF_SIZE] = { PADDING_ZERO_FILL };
00096         uint16_t encLen = 0U;
00097
00098         if (bitPadding) {
00099             /* Round up to next block boundary after appending the 0x80 marker. */
00100             uint32_t paddedLen32 = ((static_cast<uint32_t>(len) + 1U + (AES_BLOCK_SIZE_BYTES - 1U)) /
      AES_BLOCK_SIZE_BYTES) * AES_BLOCK_SIZE_BYTES;
00101             uint16_t paddedLen   = static_cast<uint16_t>(paddedLen32);
00102
00103             if (paddedLen <= static_cast<uint16_t>(AES_PAD_BUF_SIZE)) {
00104                 (void)CW_Utils::safe_memcpy(padBuf, sizeof(padBuf), in, static_cast<size_t>(len));
00105                 padBuf[static_cast<size_t>(len)] = BIT_PADDING_MARKER;
00106                 /* Remaining bytes already zero from initialisation. */
00107                 encLen = paddedLen;
00108             }
00109         } else {
00110             if (len <= static_cast<uint16_t>(AES_PAD_BUF_SIZE)) {
00111                 (void)CW_Utils::safe_memcpy(padBuf, sizeof(padBuf), in, static_cast<size_t>(len));
00112                 encLen = len;
00113             }
```

```
00114         }
00115
00116         if (encLen > 0U) {
00117             mbedtls_aes_context ctx = {};
00118             mbedtls_aes_init(&ctx);
00119
00120             unsigned int keyBits = static_cast<unsigned int>(
00121                 static_cast<uint32_t>(keyLen) * static_cast<uint32_t>(AES_KEY_BITS_PER_BYTE));
00122
00123             int ret = mbedtls_aes_setkey_enc(&ctx, key, keyBits);
00124
00125             if (ret == MBEDTLS_OK) {
00126                 ret = mbedtls_aes_crypt_cbc(&ctx, MBEDTLS_AES_ENCRYPT,
00127                                             static_cast<size_t>(encLen),
00128                                             iv, padBuf, out);
00129             }
00130
00131             mbedtls_aes_free(&ctx);
00132
00133             if (ret == MBEDTLS_OK) {
00134                 result = encLen;
00135             }
00136         }
00137     }
00138
00139     return result;
00140 }
00141
00142 /*****************************************************************
00143  * 4. AES-CBC decrypt
00144  *****************************************************************/
00145
00147 uint16_t ESP32CryptoProvider::aesCbcDecrypt(uint8_t* in, uint16_t len, uint8_t* out,
00148                                             const uint8_t* key, uint8_t keyLen,
00149                                             uint8_t* iv, bool bitPadding) {
00150     uint16_t result = 0U;
00151
00152     if ((in != NULL) && (out != NULL) && (key != NULL) && (iv != NULL) && (len > 0U)) {
00153         bool blockAligned = ((static_cast<uint32_t>(len) % AES_BLOCK_SIZE_BYTES) == 0U);
00154
00155         if (blockAligned) {
00156             mbedtls_aes_context ctx = {};
00157             mbedtls_aes_init(&ctx);
00158
00159             unsigned int keyBits = static_cast<unsigned int>(
00160                 static_cast<uint32_t>(keyLen) * static_cast<uint32_t>(AES_KEY_BITS_PER_BYTE));
00161
00162             int ret = mbedtls_aes_setkey_dec(&ctx, key, keyBits);
00163
00164             if (ret == MBEDTLS_OK) {
00165                 ret = mbedtls_aes_crypt_cbc(&ctx, MBEDTLS_AES_DECRYPT,
00166                                             static_cast<size_t>(len),
00167                                             iv, in, out);
00168             }
00169
00170             mbedtls_aes_free(&ctx);
00171
00172             if (ret == MBEDTLS_OK) {
00173                 if (bitPadding) {
00174                     /* Strip ISO/IEC 9797-1 Method 2 padding: scan backward,
00175                      * skip 0x00 bytes, then expect the 0x80 marker byte.
00176                      * padPos ends at the index of the 0x80 byte on success. */
00177                     uint16_t padPos    = len;
00178                     bool     found     = false;
00179                     bool     searching = true;
00180
00181                     while ((padPos != static_cast<uint16_t>(0U)) && searching) {
00182                         padPos--;
00183                         if (out[padPos] == BIT_PADDING_MARKER) {
00184                             found     = true;
00185                             searching = false;
00186                         } else if (out[padPos] != PADDING_ZERO_FILL) {
00187                             searching = false;
00188                         } else {
00189                             /* byte is 0x00 -- continue scanning toward the marker */
00190                         }
00191                     }
00192
00193                     if (found) {
00194                         result = padPos;  /* bytes 0 .. padPos-1 are the plaintext */
00195                     }
00196                 } else {
00197                     result = len;
00198                 }
00199             }
00200         }
00201     }
```

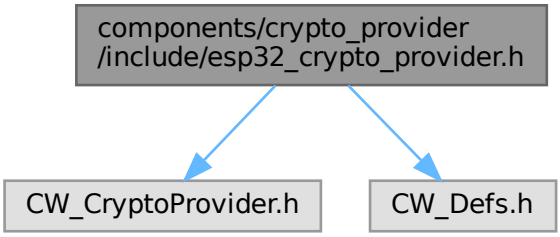
```
00202
00203     return result;
00204 }
00205
00206 /*****************************************************************
00207  * 5. ECDH and key generation (delegated to uECC shim)
00208  *****************************************************************/
00209
00211 bool ESP32CryptoProvider::ecdh(const uint8_t* pubKey, const uint8_t* privKey,
00212                                uint8_t* secret, CW_Curve curve) {
00213     bool result = false;
00214     const uECC_Curve_t* ueccCurve = toCurve(curve);
00215     if (ueccCurve != NULL) {
00216         int ret = uECC_shared_secret(pubKey, privKey, secret, ueccCurve);
00217         result = (ret == UECC_SUCCESS);
00218     }
00219     return result;
00220 }
00221
00223 bool ESP32CryptoProvider::makeKey(uint8_t* pubKey, uint8_t* privKey,
00224                                   CW_Curve curve) {
00225     bool result = false;
00226     const uECC_Curve_t* ueccCurve = toCurve(curve);
00227     if (ueccCurve != NULL) {
00228         int ret = uECC_make_key(pubKey, privKey, ueccCurve);
00229         result = (ret == UECC_SUCCESS);
00230     }
00231     return result;
00232 }
00233
00234 /*****************************************************************
00235  * 6. Random bytes from ESP32 hardware True RNG
00236  *****************************************************************/
00237
00239 bool ESP32CryptoProvider::random(uint8_t* dest, unsigned size) {
00240     bool result = false;
00241
00242     if ((dest != NULL) && (size > 0U)) {
00243         result = CW_Utils::fill_secure_random(dest, static_cast<size_t>(size));
00244     }
00245
00246     return result;
00247 }
00248
00250 bool ESP32CryptoProvider::ecdsaVerify(const uint8_t* pubKey64, const uint8_t* hash,
00251                                       size_t hashLen, const uint8_t* sig,
00252                                       CW_Curve curve) {
00253     bool result = false;
00254     const uECC_Curve_t* ueccCurve = toCurve(curve);
00255     if (ueccCurve != NULL) {
00256         int ret = uECC_verify(pubKey64, hash, static_cast<unsigned>(hashLen),
00257                               sig, ueccCurve);
00258         result = (ret == UECC_SUCCESS);
00259     }
00260     return result;
00261 }
```

5.5 components/crypto_provider/include/esp32_crypto_provider.h File Reference

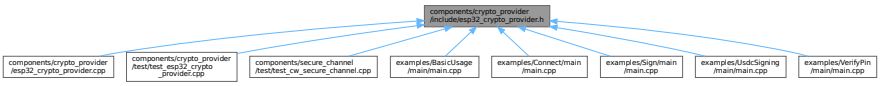
CW_CryptoProvider implementation for ESP32 using mbedTLS and the hardware TRNG.

```
#include "CW_CryptoProvider.h"
#include "CW_Defs.h"
```

Include dependency graph for esp32_crypto_provider.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ESP32CryptoProvider

  CW_CryptoProvider backed by mbedTLS and the ESP32 hardware TRNG.

5.5.1 Detailed Description

CW_CryptoProvider implementation for ESP32 using mbedTLS and the hardware TRNG. ESP32CryptoProvider wires the platform-independent CW_CryptoProvider interface to the ESP-IDF cryptographic stack:

Operation	Backend
SHA-256 / SHA-512	mbedTLS (hardware-accelerated on ESP32-S3)
AES-CBC enc / dec	mbedTLS (hardware-accelerated on ESP32-S3)
ECDH / key-gen	uECC shim backed by mbedTLS ECP primitives
ECDSA verify	mbedTLS ECP verify
Random bytes	`esp_fill_random()` — hardware TRNG (SEC-001)

Warning

The ESP32 hardware TRNG delivers full entropy only when Wi-Fi or Bluetooth is active. When neither radio is running the TRNG falls back to thermal noise and ring-oscillator jitter, which provides reduced (but non-zero) entropy. Enable Wi-Fi before performing key generation or signing operations in production firmware.

Definition in file esp32_crypto_provider.h.

5.6 esp32_crypto_provider.h

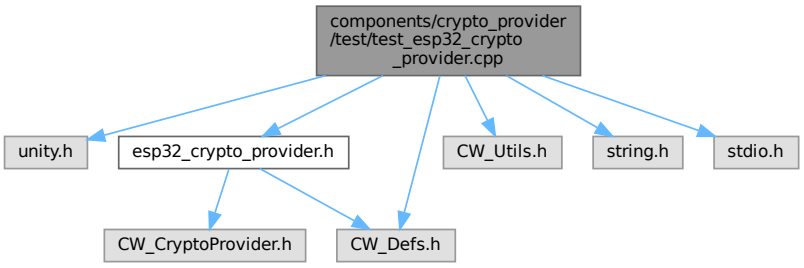
Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00029
00030 #ifndef ESP32_CRYPTO_PROVIDER_H
00031 #define ESP32_CRYPTO_PROVIDER_H
00032
00033 #include "CW_CryptoProvider.h"
00034 #include "CW_Defs.h"
00035
00062 class ESP32CryptoProvider : public CW_CryptoProvider {
00063 public:
00076     bool sha256(const uint8_t* data, size_t len, uint8_t* out) override;
00077
00090     bool sha512(const uint8_t* data, size_t len, uint8_t* out) override;
00091
00111     uint16_t aesCbcEncrypt(const uint8_t* in, uint16_t len, uint8_t* out,
00112                            const uint8_t* key, uint8_t keyLen,
00113                            uint8_t* iv, bool bitPadding) override;
00114
00133     uint16_t aesCbcDecrypt(uint8_t* in, uint16_t len, uint8_t* out,
00134                            const uint8_t* key, uint8_t keyLen,
00135                            uint8_t* iv, bool bitPadding) override;
00136
00152     bool ecdh(const uint8_t* pubKey, const uint8_t* privKey,
00153               uint8_t* secret, CW_Curve curve) override;
00154
00171     bool makeKey(uint8_t* pubKey, uint8_t* privKey,
00172                  CW_Curve curve) override;
00173
00190     bool random(uint8_t* dest, unsigned size) override;
00191
00208     bool ecdsaVerify(const uint8_t* pubKey64, const uint8_t* hash,
00209                      size_t hashLen, const uint8_t* sig,
00210                      CW_Curve curve) override;
00211
00213     ~ESP32CryptoProvider() override {}
00214 };
00215
00216 #endif /* ESP32_CRYPTO_PROVIDER_H */
```

5.7 components/crypto_provider/test/test_esp32_crypto_provider.cpp File Reference

```
#include "unity.h"
#include "esp32_crypto_provider.h"
#include "CW_Defs.h"
#include "CW_Utils.h"
#include <string.h>
#include <stdio.h>
```

Include dependency graph for test_esp32_crypto_provider.cpp:



Macros

- #define TV_SHA256_OUT_BYTES (32U)
- #define TV_SHA512_OUT_BYTES (64U)
- #define TV_AES_KEY_BYTES (16U)
- #define TV_AES_IV_BYTES (16U)
- #define TV_AES_BLOCK_BYTES (16U)
- #define TV_EC_COORD_BYTES (32U)
- #define TV_EC_PUBKEY_BYTES (64U)
- #define TV_RANDOM_BYTES (32U)
- #define TV_BIT_PAD_INPUT_BYTES (3U) /∗ plaintext shorter than one AES block ∗/

Functions

- TEST_CASE ("sha256 NIST abc vector", "[crypto_provider]")
- TEST_CASE ("sha512 NIST abc vector", "[crypto_provider]")
- TEST_CASE ("aesCbcEncrypt NIST SP800-38A F.2.1 one block", "[crypto_provider]")
- TEST_CASE ("aesCbcDecrypt NIST SP800-38A F.2.2 one block", "[crypto_provider]")
- TEST_CASE ("aesCbc bit-padding round-trip", "[crypto_provider]")
- TEST_CASE ("ecdh shared secret symmetry secp256r1", "[crypto_provider]")
- TEST_CASE ("random returns true and produces distinct draws", "[crypto_provider]")

Variables

- static ESP32CryptoProvider s_provider

5.7.1 Macro Definition Documentation

5.7.1.1 TV_AES_BLOCK_BYTES

```
#define TV_AES_BLOCK_BYTES (16U)
```

Definition at line 21 of file test_esp32_crypto_provider.cpp.
Referenced by TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.7.1.2 TV_AES_IV_BYTES

```
#define TV_AES_IV_BYTES (16U)
```

Definition at line 20 of file test_esp32_crypto_provider.cpp.
Referenced by TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.7.1.3 TV_AES_KEY_BYTES

```
#define TV_AES_KEY_BYTES (16U)
```

Definition at line 19 of file test_esp32_crypto_provider.cpp.
Referenced by TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.7.1.4 TV_BIT_PAD_INPUT_BYTES

```
#define TV_BIT_PAD_INPUT_BYTES (3U) /* plaintext shorter than one AES block */
```

Definition at line 25 of file test_esp32_crypto_provider.cpp.
Referenced by TEST_CASE().

5.7.1.5 TV_EC_COORD_BYTES

```
#define TV_EC_COORD_BYTES (32U)
```

Definition at line 22 of file test_esp32_crypto_provider.cpp.
Referenced by TEST_CASE().

5.7.1.6 TV_EC_PUBKEY_BYTES

```
#define TV_EC_PUBKEY_BYTES (64U)
```
Definition at line 23 of file test_esp32_crypto_provider.cpp.
Referenced by TEST_CASE().

5.7.1.7 TV_RANDOM_BYTES

```
#define TV_RANDOM_BYTES (32U)
```
Definition at line 24 of file test_esp32_crypto_provider.cpp.
Referenced by TEST_CASE().

5.7.1.8 TV_SHA256_OUT_BYTES

```
#define TV_SHA256_OUT_BYTES (32U)
```
Definition at line 17 of file test_esp32_crypto_provider.cpp.
Referenced by TEST_CASE().

5.7.1.9 TV_SHA512_OUT_BYTES

```
#define TV_SHA512_OUT_BYTES (64U)
```
Definition at line 18 of file test_esp32_crypto_provider.cpp.
Referenced by TEST_CASE().

5.7.2 Function Documentation

5.7.2.1 TEST_CASE() [1/7]

```
TEST_CASE (
            "aesCbc bit-padding round-trip" ,
            "" [crypto_provider])
```
Definition at line 165 of file test_esp32_crypto_provider.cpp.
References s_provider, TV_AES_BLOCK_BYTES, TV_AES_IV_BYTES, TV_AES_KEY_BYTES, and TV_BIT_PAD_INPUT_BYTES.

5.7.2.2 TEST_CASE() [2/7]

```
TEST_CASE (
            "aesCbcDecrypt NIST SP800-38A F.2.2 one block" ,
            "" [crypto_provider])
```
Definition at line 126 of file test_esp32_crypto_provider.cpp.
References s_provider, TV_AES_BLOCK_BYTES, TV_AES_IV_BYTES, and TV_AES_KEY_BYTES.

5.7.2.3 TEST_CASE() [3/7]

```
TEST_CASE (
            "aesCbcEncrypt NIST SP800-38A F.2.1 one block" ,
            "" [crypto_provider])
```
Definition at line 89 of file test_esp32_crypto_provider.cpp.
References s_provider, TV_AES_BLOCK_BYTES, TV_AES_IV_BYTES, and TV_AES_KEY_BYTES.

5.7.2.4 TEST_CASE() [4/7]

```
TEST_CASE (
            "ecdh shared secret symmetry secp256r1" ,
            "" [crypto_provider])
```
Definition at line 215 of file test_esp32_crypto_provider.cpp.
References s_provider, TV_EC_COORD_BYTES, and TV_EC_PUBKEY_BYTES.

5.7.2.5 TEST_CASE() [5/7]

```
TEST_CASE (
            "random returns true and produces distinct draws" ,
            "" [crypto_provider])
```

Definition at line 244 of file test_esp32_crypto_provider.cpp.
References s_provider, and TV_RANDOM_BYTES.

5.7.2.6 TEST_CASE() [6/7]

```
TEST_CASE (
            "sha256 NIST abc vector" ,
            "" [crypto_provider])
```

Definition at line 37 of file test_esp32_crypto_provider.cpp.
References s_provider, and TV_SHA256_OUT_BYTES.

5.7.2.7 TEST_CASE() [7/7]

```
TEST_CASE (
            "sha512 NIST abc vector" ,
            "" [crypto_provider])
```

Definition at line 65 of file test_esp32_crypto_provider.cpp.
References s_provider, and TV_SHA512_OUT_BYTES.

5.7.3 Variable Documentation

5.7.3.1 s_provider

```
ESP32CryptoProvider s_provider  [static]
```

Definition at line 31 of file test_esp32_crypto_provider.cpp.
Referenced by TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.8 test_esp32_crypto_provider.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #include "unity.h"
00007 #include "esp32_crypto_provider.h"
00008 #include "CW_Defs.h"
00009 #include "CW_Utils.h"
00010 #include <string.h>
00011 #include <stdio.h>
00012
00013 /*****************************************************************
00014  * 1. Sizes used across all test cases
00015  *****************************************************************/
00016
00017 #define TV_SHA256_OUT_BYTES    (32U)
00018 #define TV_SHA512_OUT_BYTES    (64U)
00019 #define TV_AES_KEY_BYTES       (16U)
00020 #define TV_AES_IV_BYTES        (16U)
00021 #define TV_AES_BLOCK_BYTES     (16U)
00022 #define TV_EC_COORD_BYTES      (32U)
00023 #define TV_EC_PUBKEY_BYTES     (64U)
00024 #define TV_RANDOM_BYTES        (32U)
00025 #define TV_BIT_PAD_INPUT_BYTES  (3U)  /* plaintext shorter than one AES block */
00026
00027 /*****************************************************************
00028  * 2. Static provider instance (default-constructed, no heap)
00029  *****************************************************************/
00030
00031 static ESP32CryptoProvider s_provider;
00032
00033 /*****************************************************************
00034  * 3. SHA-256 -- NIST FIPS 180-4, message "abc"
```

```
00035  *****************************************************************/
00036
00037 TEST_CASE("sha256 NIST abc vector", "[crypto_provider]")
00038 {
00039     static const uint8_t input[] = { 'a', 'b', 'c' };
00040     /* NIST FIPS 180-4 SHA-256("abc") = ba7816bf8f01cfea414140de5dae2ec7
00041      *                                  3d380fb00bbb35e2b9f27d3c0eaed7c3 */
00042     static const uint8_t expected[TV_SHA256_OUT_BYTES] = {
00043         0xbaU, 0x78U, 0x16U, 0xbfU, 0x8fU, 0x01U, 0xcfU, 0xeaU,
00044         0x41U, 0x41U, 0x40U, 0xdeU, 0x5dU, 0xaeU, 0x2eU, 0xc7U,
00045         0x3dU, 0x38U, 0x0fU, 0xb0U, 0x0bU, 0xbbU, 0x35U, 0xe2U,
00046         0xb9U, 0xf2U, 0x7dU, 0x3cU, 0x0eU, 0xaeU, 0xd7U, 0xc3U
00047     };
00048     uint8_t out[TV_SHA256_OUT_BYTES] = { 0U };
00049
00050     s_provider.sha256(input, sizeof(input), out);
00051
00052     printf("[sha256 via provider] ");
00053     for (size_t i = 0U; i < TV_SHA256_OUT_BYTES; i++) {
00054         printf("%02x", static_cast<unsigned int>(out[i]));
00055     }
00056     printf("\n");
00057
00058     TEST_ASSERT_EQUAL_HEX8_ARRAY(expected, out, TV_SHA256_OUT_BYTES);
00059 }
00060
00061 /*****************************************************************
00062  * 4. SHA-512 -- NIST FIPS 180-4, message "abc"
00063  *****************************************************************/
00064
00065 TEST_CASE("sha512 NIST abc vector", "[crypto_provider]")
00066 {
00067     static const uint8_t input[] = { 'a', 'b', 'c' };
00068     static const uint8_t expected[TV_SHA512_OUT_BYTES] = {
00069         0xddU, 0xafU, 0x35U, 0xa1U, 0x93U, 0x61U, 0x7aU, 0xbaU,
00070         0xccU, 0x41U, 0x73U, 0x49U, 0xaeU, 0x20U, 0x41U, 0x31U,
00071         0x12U, 0xe6U, 0xfaU, 0x4eU, 0x89U, 0xa9U, 0x7eU, 0xa2U,
00072         0x0aU, 0x9eU, 0xeeU, 0xe6U, 0x4bU, 0x55U, 0xd3U, 0x9aU,
00073         0x21U, 0x92U, 0x99U, 0x2aU, 0x27U, 0x4fU, 0xc1U, 0xa8U,
00074         0x36U, 0xbaU, 0x3cU, 0x23U, 0xa3U, 0xfeU, 0xebU, 0xbdU,
00075         0x45U, 0x4dU, 0x44U, 0x23U, 0x64U, 0x3cU, 0xe8U, 0x0eU,
00076         0x2aU, 0x9aU, 0xc9U, 0x4fU, 0xa5U, 0x4cU, 0xa4U, 0x9fU
00077     };
00078     uint8_t out[TV_SHA512_OUT_BYTES] = { 0U };
00079
00080     s_provider.sha512(input, sizeof(input), out);
00081
00082     TEST_ASSERT_EQUAL_HEX8_ARRAY(expected, out, TV_SHA512_OUT_BYTES);
00083 }
00084
00085 /*****************************************************************
00086  * 5. AES-128-CBC encrypt -- NIST SP 800-38A F.2.1, one block
00087  *****************************************************************/
00088
00089 TEST_CASE("aesCbcEncrypt NIST SP800-38A F.2.1 one block", "[crypto_provider]")
00090 {
00091     static const uint8_t key[TV_AES_KEY_BYTES] = {
00092         0x2bU, 0x7eU, 0x15U, 0x16U, 0x28U, 0xaeU, 0xd2U, 0xa6U,
00093         0xabU, 0xf7U, 0x15U, 0x88U, 0x09U, 0xcfU, 0x4fU, 0x3cU
00094     };
00095     static const uint8_t plaintext[TV_AES_BLOCK_BYTES] = {
00096         0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
00097         0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU
00098     };
00099     static const uint8_t expected[TV_AES_BLOCK_BYTES] = {
00100         0x76U, 0x49U, 0xabU, 0xacU, 0x81U, 0x19U, 0xb2U, 0x46U,
00101         0xceU, 0xe9U, 0x8eU, 0x9bU, 0x12U, 0xe9U, 0x19U, 0x7dU
00102     };
00103     uint8_t iv[TV_AES_IV_BYTES] = {
00104         0x00U, 0x01U, 0x02U, 0x03U, 0x04U, 0x05U, 0x06U, 0x07U,
00105         0x08U, 0x09U, 0x0aU, 0x0bU, 0x0cU, 0x0dU, 0x0eU, 0x0fU
00106     };
00107     uint8_t  out[TV_AES_BLOCK_BYTES] = { 0U };
00108
00109     uint16_t encLen = s_provider.aesCbcEncrypt(
00110         plaintext,
00111         static_cast<uint16_t>(sizeof(plaintext)),
00112         out,
00113         key,
00114         static_cast<uint8_t>(sizeof(key)),
00115         iv,
00116         false);
00117
00118     TEST_ASSERT_EQUAL_UINT16(static_cast<uint16_t>(TV_AES_BLOCK_BYTES), encLen);
00119     TEST_ASSERT_EQUAL_HEX8_ARRAY(expected, out, TV_AES_BLOCK_BYTES);
00120 }
00121
```

```
00122 /*****************************************************************
00123  * 6. AES-128-CBC decrypt -- NIST SP 800-38A F.2.2, one block
00124  *****************************************************************/
00125
00126 TEST_CASE("aesCbcDecrypt NIST SP800-38A F.2.2 one block", "[crypto_provider]")
00127 {
00128     static const uint8_t key[TV_AES_KEY_BYTES] = {
00129         0x2bU, 0x7eU, 0x15U, 0x16U, 0x28U, 0xaeU, 0xd2U, 0xa6U,
00130         0xabU, 0xf7U, 0x15U, 0x88U, 0x09U, 0xcfU, 0x4fU, 0x3cU
00131     };
00132     uint8_t ciphertext[TV_AES_BLOCK_BYTES] = {
00133         0x76U, 0x49U, 0xabU, 0xacU, 0x81U, 0x19U, 0xb2U, 0x46U,
00134         0xceU, 0xe9U, 0x8eU, 0x9bU, 0x12U, 0xe9U, 0x19U, 0x7dU
00135     };
00136     static const uint8_t expected[TV_AES_BLOCK_BYTES] = {
00137         0x6bU, 0xc1U, 0xbeU, 0xe2U, 0x2eU, 0x40U, 0x9fU, 0x96U,
00138         0xe9U, 0x3dU, 0x7eU, 0x11U, 0x73U, 0x93U, 0x17U, 0x2aU
00139     };
00140     uint8_t iv[TV_AES_IV_BYTES] = {
00141         0x00U, 0x01U, 0x02U, 0x03U, 0x04U, 0x05U, 0x06U, 0x07U,
00142         0x08U, 0x09U, 0x0aU, 0x0bU, 0x0cU, 0x0dU, 0x0eU, 0x0fU
00143     };
00144     uint8_t  out[TV_AES_BLOCK_BYTES] = { 0U };
00145
00146     uint16_t decLen = s_provider.aesCbcDecrypt(
00147         ciphertext,
00148         static_cast<uint16_t>(sizeof(ciphertext)),
00149         out,
00150         key,
00151         static_cast<uint8_t>(sizeof(key)),
00152         iv,
00153         false);
00154
00155     TEST_ASSERT_EQUAL_UINT16(static_cast<uint16_t>(TV_AES_BLOCK_BYTES), decLen);
00156     TEST_ASSERT_EQUAL_HEX8_ARRAY(expected, out, TV_AES_BLOCK_BYTES);
00157 }
00158
00159 /*****************************************************************
00160  * 7. AES-128-CBC bit-padding round-trip
00161  *    Input shorter than one block → padded to 16 bytes on encrypt,
00162  *    stripped back to original length on decrypt.
00163  *****************************************************************/
00164
00165 TEST_CASE("aesCbc bit-padding round-trip", "[crypto_provider]")
00166 {
00167     static const uint8_t key[TV_AES_KEY_BYTES] = {
00168         0x2bU, 0x7eU, 0x15U, 0x16U, 0x28U, 0xaeU, 0xd2U, 0xa6U,
00169         0xabU, 0xf7U, 0x15U, 0x88U, 0x09U, 0xcfU, 0x4fU, 0x3cU
00170     };
00171     static const uint8_t plaintext[TV_BIT_PAD_INPUT_BYTES] = {
00172         0x01U, 0x02U, 0x03U
00173     };
00174
00175     uint8_t iv_enc[TV_AES_IV_BYTES] = {
00176         0x00U, 0x01U, 0x02U, 0x03U, 0x04U, 0x05U, 0x06U, 0x07U,
00177         0x08U, 0x09U, 0x0aU, 0x0bU, 0x0cU, 0x0dU, 0x0eU, 0x0fU
00178     };
00179     uint8_t iv_dec[TV_AES_IV_BYTES] = {
00180         0x00U, 0x01U, 0x02U, 0x03U, 0x04U, 0x05U, 0x06U, 0x07U,
00181         0x08U, 0x09U, 0x0aU, 0x0bU, 0x0cU, 0x0dU, 0x0eU, 0x0fU
00182     };
00183     uint8_t ciphertext[TV_AES_BLOCK_BYTES] = { 0U };
00184     uint8_t recovered[TV_AES_BLOCK_BYTES]  = { 0U };
00185
00186     uint16_t encLen = s_provider.aesCbcEncrypt(
00187         plaintext,
00188         static_cast<uint16_t>(sizeof(plaintext)),
00189         ciphertext,
00190         key,
00191         static_cast<uint8_t>(sizeof(key)),
00192         iv_enc,
00193         true);
00194
00195     /* 3 bytes + 0x80 marker → padded to one full block */
00196     TEST_ASSERT_EQUAL_UINT16(static_cast<uint16_t>(TV_AES_BLOCK_BYTES), encLen);
00197
00198     uint16_t decLen = s_provider.aesCbcDecrypt(
00199         ciphertext,
00200         encLen,
00201         recovered,
00202         key,
00203         static_cast<uint8_t>(sizeof(key)),
00204         iv_dec,
00205         true);
00206
00207     TEST_ASSERT_EQUAL_UINT16(static_cast<uint16_t>(TV_BIT_PAD_INPUT_BYTES), decLen);
00208     TEST_ASSERT_EQUAL_HEX8_ARRAY(plaintext, recovered, TV_BIT_PAD_INPUT_BYTES);
```

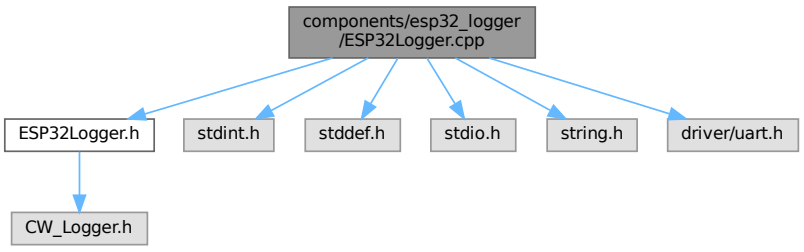
```
00209 }
00210
00211 /*****************************************************************
00212  * 8. ECDH -- two-party shared secret symmetry on secp256r1
00213  *****************************************************************/
00214
00215 TEST_CASE("ecdh shared secret symmetry secp256r1", "[crypto_provider]")
00216 {
00217     uint8_t pubA[TV_EC_PUBKEY_BYTES]   = { 0U };
00218     uint8_t privA[TV_EC_COORD_BYTES]   = { 0U };
00219     uint8_t pubB[TV_EC_PUBKEY_BYTES]   = { 0U };
00220     uint8_t privB[TV_EC_COORD_BYTES]   = { 0U };
00221     uint8_t secretA[TV_EC_COORD_BYTES] = { 0U };
00222     uint8_t secretB[TV_EC_COORD_BYTES] = { 0U };
00223
00224     CW_Curve curve = CW_CURVE_SECP256R1;
00225
00226     bool okA = s_provider.makeKey(pubA, privA, curve);
00227     bool okB = s_provider.makeKey(pubB, privB, curve);
00228
00229     TEST_ASSERT_TRUE(okA);
00230     TEST_ASSERT_TRUE(okB);
00231
00232     bool ecdhA = s_provider.ecdh(pubB, privA, secretA, curve);
00233     bool ecdhB = s_provider.ecdh(pubA, privB, secretB, curve);
00234
00235     TEST_ASSERT_TRUE(ecdhA);
00236     TEST_ASSERT_TRUE(ecdhB);
00237     TEST_ASSERT_EQUAL_HEX8_ARRAY(secretA, secretB, TV_EC_COORD_BYTES);
00238 }
00239
00240 /*****************************************************************
00241  * 9. random -- returns true; two draws from the TRNG must differ
00242  *****************************************************************/
00243
00244 TEST_CASE("random returns true and produces distinct draws", "[crypto_provider]")
00245 {
00246     uint8_t buf1[TV_RANDOM_BYTES] = { 0U };
00247     uint8_t buf2[TV_RANDOM_BYTES] = { 0U };
00248
00249     bool randomGeneration1Succeeded = s_provider.random(buf1, TV_RANDOM_BYTES);
00250     bool randomGeneration2Succeeded = s_provider.random(buf2, TV_RANDOM_BYTES);
00251
00252     TEST_ASSERT_TRUE(randomGeneration1Succeeded);
00253     TEST_ASSERT_TRUE(randomGeneration2Succeeded);
00254     /* P(collision of two independent 32-byte TRNG draws) < 2^-256. */
00255     TEST_ASSERT_FALSE(CW_Utils::secure_compare(buf1, buf2, TV_RANDOM_BYTES));
00256 }
```

5.9 components/esp32_logger/ESP32Logger.cpp File Reference

Implementation of ESP32Logger — ESP32 UART0 logging backend.

```
#include "ESP32Logger.h"
#include <stdint.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include "driver/uart.h"
```

Include dependency graph for ESP32Logger.cpp:



Functions

- static uint32_t clamp_base (int base)
- static void uart_write_str (const char ∗str)
- static void write_uint_to_uart (uint32_t value, uint32_t base)

Variables

- static const uart_port_t UART_LOG_PORT = UART_NUM_0
- static const uint8_t UART_RX_FLOW_THRESH_NONE = static_cast<uint8_t>(0U)
- static const uint32_t NUM_BUF_SIZE = 33U
- static const uint32_t NUM_BASE_MIN = 2U
- static const uint32_t NUM_BASE_MAX = 16U
- static const size_t ELEMENT_SIZE = 1U
- static const char LOGGER_NEWLINE [ ] = "\r\n"
- static const char HEX_CHARS [ ] = "0123456789ABCDEF"

5.9.1 Detailed Description

Implementation of ESP32Logger — ESP32 UART0 logging backend.
Routes all CW_Logger output through the ESP-IDF UART driver on UART0. Full API documentation lives on the declarations in ESP32Logger.h.

Warning

Do not ship this logger in production firmware; it exposes APDU traces and PIN values on the serial console (LOW-03).

Definition in file ESP32Logger.cpp.

5.9.2 Function Documentation

5.9.2.1 clamp_base()

```
uint32_t clamp_base (
            int base)  [static]
```

Clamp base to the valid [NUM_BASE_MIN, NUM_BASE_MAX] range; returns DEC on out-of-range input.
Definition at line 50 of file ESP32Logger.cpp.
References NUM_BASE_MAX, and NUM_BASE_MIN.
Referenced by ESP32Logger::print(), ESP32Logger::print(), ESP32Logger::print(), and ESP32Logger::print().

5.9.2.2 uart_write_str()

```
void uart_write_str (
            const char * str)  [static]
```

Transmit a NUL-terminated string to stdout via fwrite.
Definition at line 62 of file ESP32Logger.cpp.
References ELEMENT_SIZE.
Referenced by ESP32Logger::print(), ESP32Logger::print(), and ESP32Logger::println().

5.9.2.3 write_uint_to_uart()

```
void write_uint_to_uart (
            uint32_t value,
            uint32_t base)  [static]
```

Convert value to ASCII digits in the given base and write the result to stdout.
Definition at line 69 of file ESP32Logger.cpp.
References HEX_CHARS, and NUM_BUF_SIZE.
Referenced by ESP32Logger::print(), ESP32Logger::print(), ESP32Logger::print(), and ESP32Logger::print().

5.9.3 Variable Documentation

5.9.3.1 ELEMENT_SIZE

```
const size_t ELEMENT_SIZE = 1U  [static]
```
Definition at line 40 of file ESP32Logger.cpp.
Referenced by uart_write_str().

5.9.3.2 HEX_CHARS

```
const char HEX_CHARS[] = "0123456789ABCDEF"  [static]
```
Definition at line 43 of file ESP32Logger.cpp.
Referenced by write_uint_to_uart().

5.9.3.3 LOGGER_NEWLINE

```
const char LOGGER_NEWLINE[] = "\r\n"  [static]
```
Definition at line 42 of file ESP32Logger.cpp.
Referenced by ESP32Logger::println().

5.9.3.4 NUM_BASE_MAX

```
const uint32_t NUM_BASE_MAX = 16U  [static]
```
Definition at line 39 of file ESP32Logger.cpp.
Referenced by clamp_base().

5.9.3.5 NUM_BASE_MIN

```
const uint32_t NUM_BASE_MIN = 2U  [static]
```
Definition at line 38 of file ESP32Logger.cpp.
Referenced by clamp_base().

5.9.3.6 NUM_BUF_SIZE

```
const uint32_t NUM_BUF_SIZE = 33U  [static]
```
Definition at line 37 of file ESP32Logger.cpp.
Referenced by write_uint_to_uart().

5.9.3.7 UART_LOG_PORT

```
const uart_port_t UART_LOG_PORT = UART_NUM_0  [static]
```
Definition at line 33 of file ESP32Logger.cpp.
Referenced by ESP32Logger::begin().

5.9.3.8 UART_RX_FLOW_THRESH_NONE

```
const uint8_t UART_RX_FLOW_THRESH_NONE = static_cast<uint8_t>(0U)  [static]
```
Definition at line 34 of file ESP32Logger.cpp.
Referenced by ESP32Logger::begin().

5.10 ESP32Logger.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00016
```

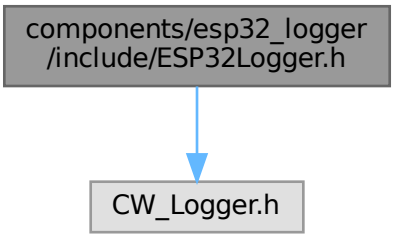
```
00017 /*****************************************************************
00018  * 1. Included files
00019  *****************************************************************/
00020
00021 #include "ESP32Logger.h"
00022
00023 #include <stdint.h>
00024 #include <stddef.h>
00025 #include <stdio.h>
00026 #include <string.h>
00027 #include "driver/uart.h"
00028
00029 /*****************************************************************
00030  * 2. Module constants
00031  *****************************************************************/
00032
00033 static const uart_port_t UART_LOG_PORT              = UART_NUM_0;
00034 static const uint8_t     UART_RX_FLOW_THRESH_NONE   = static_cast<uint8_t>(0U);
00035
00036 /* 32 binary digits + NUL terminator */
00037 static const uint32_t NUM_BUF_SIZE  = 33U;
00038 static const uint32_t NUM_BASE_MIN  = 2U;
00039 static const uint32_t NUM_BASE_MAX  = 16U;
00040 static const size_t   ELEMENT_SIZE  = 1U;
00041
00042 static const char LOGGER_NEWLINE[] = "\r\n";
00043 static const char HEX_CHARS[]      = "0123456789ABCDEF";
00044
00045 /*****************************************************************
00046  * 3. Static helpers
00047  *****************************************************************/
00048
00050 static uint32_t clamp_base(int base)
00051 {
00052     uint32_t result = static_cast<uint32_t>(DEC);
00053     bool in_range   = ((static_cast<uint32_t>(base) >= NUM_BASE_MIN) &&
00054                        (static_cast<uint32_t>(base) <= NUM_BASE_MAX));
00055     if (in_range) {
00056         result = static_cast<uint32_t>(base);
00057     }
00058     return result;
00059 }
00060
00062 static void uart_write_str(const char *str)
00063 {
00064     size_t len = strlen(str);
00065     (void)fwrite(str, ELEMENT_SIZE, len, stdout);
00066 }
00067
00069 static void write_uint_to_uart(uint32_t value, uint32_t base)
00070 {
00071     char     buf[NUM_BUF_SIZE] = { 0 };
00072     uint32_t pos               = NUM_BUF_SIZE - 1U;
00073     uint32_t v                 = value;
00074
00075     buf[pos] = '\0';
00076
00077     if (v == 0U) {
00078         pos--;
00079         buf[pos] = '0';
00080     } else {
00081         while (v > 0U) {
00082             pos--;
00083             buf[pos] = HEX_CHARS[v % base];
00084             v        = v / base;
00085         }
00086     }
00087
00088     (void)fputs(&buf[pos], stdout);
00089 }
00090
00091 /*****************************************************************
00092  * 4. Public method implementations
00093  *****************************************************************/
00094
00095 bool ESP32Logger::begin(unsigned long baudRate)
00096 {
00097     uart_config_t cfg = {};
00098     cfg.baud_rate           = static_cast<int>(baudRate);
00099     cfg.data_bits           = UART_DATA_8_BITS;
00100     cfg.parity              = UART_PARITY_DISABLE;
00101     cfg.stop_bits           = UART_STOP_BITS_1;
00102     cfg.flow_ctrl           = UART_HW_FLOWCTRL_DISABLE;
00103     cfg.rx_flow_ctrl_thresh = UART_RX_FLOW_THRESH_NONE;
00104     cfg.source_clk          = UART_SCLK_DEFAULT;
00105
00106     esp_err_t err  = uart_param_config(UART_LOG_PORT, &cfg);
```

```
00107     m_initialized  = (err == ESP_OK);
00108
00109     return m_initialized;
00110 }
00111
00112 void ESP32Logger::print(const __FlashStringHelper *str)
00113 {
00114     if (m_initialized) {
00115         uart_write_str(reinterpret_cast<const char *>(str));
00116     }
00117 }
00118
00119 void ESP32Logger::print(const char *str)
00120 {
00121     if (m_initialized) {
00122         uart_write_str(str);
00123     }
00124 }
00125
00126 void ESP32Logger::print(char c)
00127 {
00128     if (m_initialized) {
00129         (void)fputc(static_cast<int>(c), stdout);
00130     }
00131 }
00132
00133 void ESP32Logger::print(uint8_t value, int base)
00134 {
00135     if (m_initialized) {
00136         uint32_t safe_base = clamp_base(base);
00137         write_uint_to_uart(static_cast<uint32_t>(value), safe_base);
00138     }
00139 }
00140
00141 void ESP32Logger::print(uint16_t value, int base)
00142 {
00143     if (m_initialized) {
00144         uint32_t safe_base = clamp_base(base);
00145         write_uint_to_uart(static_cast<uint32_t>(value), safe_base);
00146     }
00147 }
00148
00149 void ESP32Logger::print(uint32_t value, int base)
00150 {
00151     if (m_initialized) {
00152         uint32_t safe_base = clamp_base(base);
00153         write_uint_to_uart(value, safe_base);
00154     }
00155 }
00156
00157 void ESP32Logger::print(int value, int base)
00158 {
00159     if (m_initialized) {
00160         uint32_t safe_base               = clamp_base(base);
00161         bool     is_negative_decimal     = ((value < 0) && (safe_base == static_cast<uint32_t>(DEC)));
00162         if (is_negative_decimal) {
00163             (void)fputc(static_cast<int>('-'), stdout);
00164             /* Two's-complement negation -- avoids UB on INT_MIN */
00165             uint32_t abs_val = (~static_cast<uint32_t>(value)) + 1U;
00166             write_uint_to_uart(abs_val, safe_base);
00167         } else {
00168             write_uint_to_uart(static_cast<uint32_t>(value), safe_base);
00169         }
00170     }
00171 }
00172
00173 void ESP32Logger::println()
00174 {
00175     if (m_initialized) {
00176         uart_write_str(LOGGER_NEWLINE);
00177     }
00178 }
00179
00180 void ESP32Logger::println(const __FlashStringHelper *str)
00181 {
00182     print(str);
00183     println();
00184 }
00185
00186 void ESP32Logger::println(const char *str)
00187 {
00188     print(str);
00189     println();
00190 }
00191
00192 void ESP32Logger::println(char c)
00193 {
```

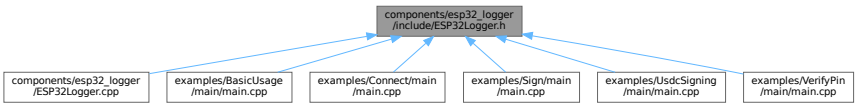
```
00194     print(c);
00195     println();
00196 }
00197
00198 void ESP32Logger::println(uint8_t value, int base)
00199 {
00200     print(value, base);
00201     println();
00202 }
00203
00204 void ESP32Logger::println(uint16_t value, int base)
00205 {
00206     print(value, base);
00207     println();
00208 }
00209
00210 void ESP32Logger::println(uint32_t value, int base)
00211 {
00212     print(value, base);
00213     println();
00214 }
00215
00216 void ESP32Logger::println(int value, int base)
00217 {
00218     print(value, base);
00219     println();
00220 }
```

5.11 components/esp32_logger/include/ESP32Logger.h File Reference

CW_Logger implementation that writes to ESP32 UART0 via `printf`.
`#include "CW_Logger.h"`
Include dependency graph for ESP32Logger.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ESP32Logger

  CW_Logger backed by ESP32 UART0.

5.11.1 Detailed Description

CW_Logger implementation that writes to ESP32 UART0 via `printf`.
ESP32Logger maps every CW_Logger virtual method to the ESP-IDF `uart_write_bytes` / `printf` family so the platform-independent SDK core can produce human-readable diagnostic output on the serial console without depending on Arduino's `Serial` object.

Usage

```
ESP32Logger logger;
logger.begin(115200UL);
CryptnoxWallet wallet(transport, logger, crypto, platform);
```

Definition in file ESP32Logger.h.

5.12 ESP32Logger.h

Go to the documentation of this file.

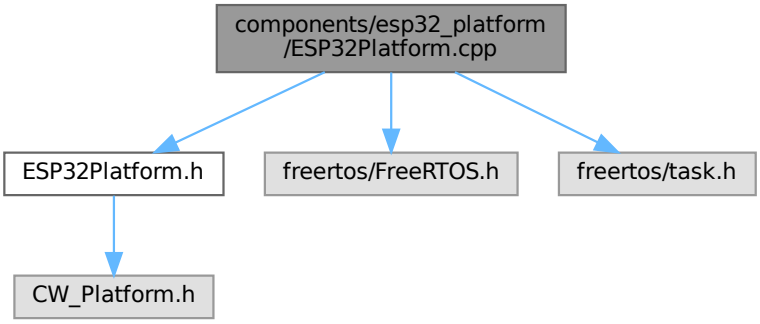
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00024
00025 #ifndef ESP32_LOGGER_H
00026 #define ESP32_LOGGER_H
00027
00028 #include "CW_Logger.h"
00029
00048 class ESP32Logger : public CW_Logger {
00049 public:
00052
00059     bool begin(unsigned long baudRate = 115200UL) override;
00060
00062
00065
00074     void print(const __FlashStringHelper *str) override;
00075
00081     void print(const char *str) override;
00082
00088     void print(char c) override;
00089
00097     void print(uint8_t  value, int base = DEC) override;
00098
00105     void print(uint16_t value, int base = DEC) override;
00106
00113     void print(uint32_t value, int base = DEC) override;
00114
00123     void print(int value, int base = DEC) override;
00124
00126
00129
00133     void println() override;
00134
00140     void println(const __FlashStringHelper *str) override;
00141
00147     void println(const char *str) override;
00148
00154     void println(char c) override;
00155
00162     void println(uint8_t  value, int base = DEC) override;
00163
00170     void println(uint16_t value, int base = DEC) override;
00171
00178     void println(uint32_t value, int base = DEC) override;
00179
00186     void println(int value, int base = DEC) override;
00187
00189
00191     ~ESP32Logger() override {}
00192
00193 private:
00194     bool m_initialized = false;
00195 };
00196
00197 #endif /* ESP32_LOGGER_H */
```

5.13 components/esp32_platform/ESP32Platform.cpp File Reference

Implementation of ESP32Platform — FreeRTOS sleep backend.

```
#include "ESP32Platform.h"
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
```

Include dependency graph for ESP32Platform.cpp:



5.13.1 Detailed Description

Implementation of ESP32Platform — FreeRTOS sleep backend.
Delegates CW_Platform::sleep_ms to `vTaskDelay` so the calling FreeRTOS task yields to the scheduler for the requested duration. Full API documentation lives on the declarations in ESP32Platform.h.
Definition in file ESP32Platform.cpp.

5.14 ESP32Platform.cpp

Go to the documentation of this file.

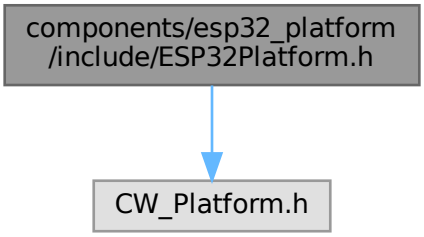
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00014
00015 #include "ESP32Platform.h"
00016 #include <freertos/FreeRTOS.h>
00017 #include <freertos/task.h>
00018
00020 void ESP32Platform::sleep_ms(uint32_t ms) {
00021     vTaskDelay(pdMS_TO_TICKS(static_cast<TickType_t>(ms)));
00022 }
```

5.15 components/esp32_platform/include/ESP32Platform.h File Reference

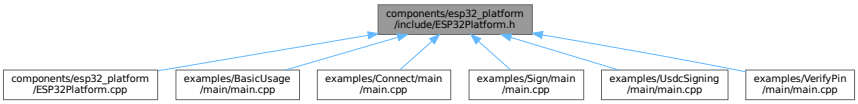
CW_Platform implementation for ESP32 using FreeRTOS.

```
#include "CW_Platform.h"
```

Include dependency graph for ESP32Platform.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ESP32Platform

  CW_Platform backed by FreeRTOS `vTaskDelay`.

5.15.1 Detailed Description

CW_Platform implementation for ESP32 using FreeRTOS.
ESP32Platform maps the single CW_Platform::sleep_ms abstraction to `vTaskDelay` so the platform-independent SDK core can insert timed pauses without depending on any Arduino or RTOS header directly.
Definition in file ESP32Platform.h.

5.16 ESP32Platform.h

Go to the documentation of this file.

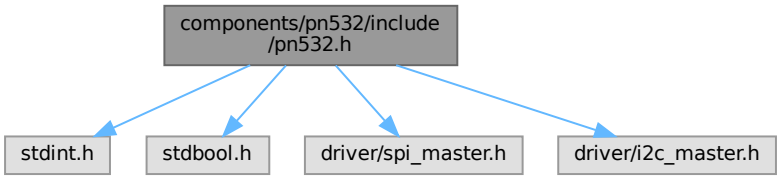
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00016
00017 #ifndef ESP32_PLATFORM_H
00018 #define ESP32_PLATFORM_H
00019
00020 #include "CW_Platform.h"
00021
00038 class ESP32Platform : public CW_Platform {
00039 public:
00049     void sleep_ms(uint32_t ms) override;
00050
00052     ~ESP32Platform() override {}
00053 };
00054
00055 #endif /* ESP32_PLATFORM_H */
```

5.17 components/pn532/include/pn532.h File Reference

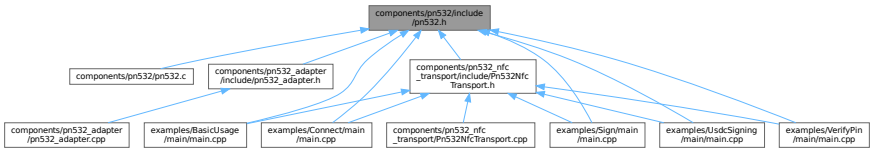
Low-level PN532 NFC controller driver for ESP-IDF (SPI and I²C).

```
#include <stdint.h>
#include <stdbool.h>
#include "driver/spi_master.h"
#include "driver/i2c_master.h"
```

Include dependency graph for pn532.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct pn532_config_t

  Compile-time configuration passed to pn532_init.
- struct pn532_t

  Opaque-like runtime state for a single PN532 instance.

Macros

- #define PN532_MIFARE_ISO14443A (0x00U)

  Baud-rate selector for ISO 14443-A (Mifare) cards passed to pn532_read_passive_target_id.
- #define PN532_MAX_APDU_LEN (252U)

  Maximum APDU payload length accepted by pn532_send_apdu.
- #define PN532_I2C_ADDRESS (0x24U)

  7-bit I²C slave address of the PN532 (fixed in hardware).

Enumerations

- enum pn532_transport_t { PN532_TRANSPORT_SPI , PN532_TRANSPORT_I2C }

  Physical bus used to communicate with the PN532.

Functions

- esp_err_t pn532_init (pn532_t *dev, const pn532_config_t *config)

  Initialise the PN532 and bring it to a ready state.
- uint32_t pn532_get_firmware_version (pn532_t *dev)

  Query the PN532 firmware version.
- bool pn532_sam_config (pn532_t *dev)

  Configure the PN532's Security Access Module (SAM).
- uint32_t pn532_read_passive_target_id (pn532_t *dev, uint8_t cardbaudrate)

  Scan for a passive ISO 14443-A card and return its UID.
- bool pn532_send_apdu (pn532_t *dev, const uint8_t *apdu, uint8_t apdu_len, uint8_t *response, uint16_t *response_len)

  Exchange a single ISO-DEP APDU with the currently selected card.
- bool pn532_release_target (pn532_t *dev)

  Release the currently selected NFC target.

5.17.1 Detailed Description

Low-level PN532 NFC controller driver for ESP-IDF (SPI and I²C).
Provides a transport-agnostic C API for the NXP PN532 NFC controller. Both the ESP-IDF SPI master peripheral and the IDF v5.x I²C master API are supported; the active transport is selected at pn532_init time via pn532_config_t::transport.
Typical usage (SPI):

```
pn532_config_t cfg = {};
cfg.transport     = PN532_TRANSPORT_SPI;
cfg.spi_host      = SPI2_HOST;
cfg.pin_cs        = 10;
cfg.skip_bus_init = true;   // caller already called spi_bus_initialize()

pn532_t nfc;
ESP_ERROR_CHECK(pn532_init(&nfc, &cfg));
```

Definition in file pn532.h.

5.18 pn532.h

Go to the documentation of this file.

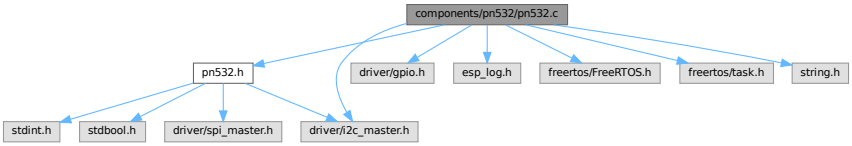
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later AND BSD-3-Clause
00003  *
00004  * Public PN532 driver header. Portions derive from Adafruit_PN532
00005  * (Copyright (c) 2012, Adafruit Industries; BSD-3-Clause). See
00006  * NOTICES.md at the repo root for the full notice.
00007  */
00008
00033
00034 #pragma once
00035
00036 #include <stdint.h>
00037 #include <stdbool.h>
00038 #include "driver/spi_master.h"
00039 #include "driver/i2c_master.h"   /* new IDF v5.x master API */
00040
00041 #ifdef __cplusplus
00042 extern "C" {
00043 #endif
00044
00048
00049 /* - Card / protocol constants ------------------------ */
00050
00053 /* cppcheck-suppress misra-c2012-2.5 */
00054 #define PN532_MIFARE_ISO14443A  (0x00U)
00055
00060 #define PN532_MAX_APDU_LEN      (252U)
00061
00063 #define PN532_I2C_ADDRESS       (0x24U)
00064
00065 /* - Transport selector ----------------------------- */
00066
00076 typedef enum {
```

```
00077     PN532_TRANSPORT_SPI,
00078     PN532_TRANSPORT_I2C,
00079 } pn532_transport_t;
00080
00081 /* - Configuration structure ------------------------ */
00082
00111 typedef struct {
00112     pn532_transport_t transport;
00113
00114     /* - SPI fields (used when transport == PN532_TRANSPORT_SPI) - */
00115     spi_host_device_t spi_host;
00116     int               pin_mosi;
00117     int               pin_miso;
00118     int               pin_sclk;
00119     int               pin_cs;
00120     bool              skip_bus_init;
00121
00122     /* - I²C fields (used when transport == PN532_TRANSPORT_I2C) - */
00123     int      i2c_port;
00124     int      pin_sda;
00125     int      pin_scl;
00126     int      pin_irq;
00127     int      pin_rst;
00128     uint32_t i2c_clock_hz;
00129 } pn532_config_t;
00130
00131 /* - Runtime device state --------------------------- */
00132
00141 typedef struct {
00142     pn532_transport_t transport;
00143
00144     /* - SPI state - */
00145     spi_device_handle_t spi;
00146     int                 pin_cs;
00147
00148     /* - I²C state - */
00149     i2c_master_bus_handle_t i2c_bus;
00150     i2c_master_dev_handle_t i2c_dev;
00151     int                     pin_irq;
00152     int                     pin_rst;
00153 } pn532_t;
00154
00155 /* - Public API ---------------------------------- */
00156
00178 esp_err_t pn532_init(pn532_t *dev, const pn532_config_t *config);
00179
00194 uint32_t pn532_get_firmware_version(pn532_t *dev);
00195
00210 bool pn532_sam_config(pn532_t *dev);
00211
00226 uint32_t pn532_read_passive_target_id(pn532_t *dev, uint8_t cardbaudrate);
00227
00250 bool pn532_send_apdu(pn532_t *dev, const uint8_t *apdu, uint8_t apdu_len,
00251                      uint8_t *response, uint16_t *response_len);
00252
00264 bool pn532_release_target(pn532_t *dev);
00265  /* end of pn532_driver group */
00267
00268 #ifdef __cplusplus
00269 }
00270 #endif
```

5.19 components/pn532/pn532.c File Reference

```
#include "pn532.h"
#include "driver/gpio.h"
#include "driver/i2c_master.h"
#include "esp_log.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include <string.h>
```

Include dependency graph for pn532.c:



Macros

- #define PN532_PREAMBLE (0x00U)
- #define PN532_STARTCODE1 (0x00U)
- #define PN532_STARTCODE2 (0xFFU)
- #define PN532_POSTAMBLE (0x00U)
- #define PN532_HOSTTOPN532 (0xD4U)
- #define PN532_FIRMWAREVERSION (0x02U)
- #define PN532_SAMCONFIGURATION (0x14U)
- #define PN532_INLISTPASSIVETARGET (0x4AU)
- #define PN532_INDATAEXCHANGE (0x40U)
- #define PN532_INRELEASE (0x52U)
- #define PN532_SPI_STATREAD (0x02U)
- #define PN532_SPI_DATAWRITE (0x01U)
- #define PN532_SPI_DATAREAD (0x03U)
- #define PN532_SPI_READY (0x01U)
- #define PN532_SPI_CLOCK_HZ (1000000U)
- #define PN532_SPI_MODE (0U)
- #define PN532_SPI_NO_PIN (-1)
- #define PN532_SPI_QUEUE_SIZE (1U)
- #define PN532_SPI_BITS (8U)
- #define PN532_I2C_TIMEOUT_MS (100)
- #define PN532_I2C_TX_MAX (PN532_MAX_APDU_LEN + 16U)
- #define PN532_I2C_RX_MAX (200U)
- #define PN532_CS_TOGGLE_DELAY_MS (2U)
- #define PN532_WAKEUP_DELAY_MS (1000U)
- #define PN532_SYNC_DELAY_MS (100U)
- #define PN532_POLL_INTERVAL_MS (10U)
- #define PN532_CMD_TIMEOUT_MS (1000U)
- #define PN532_APDU_TIMEOUT_MS (5000U)
- #define PN532_BYTE_DELAY_MS (1U)
- #define GPIO_LEVEL_LOW (0U) /∗ drive pin low ∗/
- #define GPIO_LEVEL_HIGH (1U) /∗ drive pin high ∗/
- #define GPIO_PIN_BITMASK_BASE (1ULL) /∗ base bit for gpio_config_t.pin_bit_mask ∗/
- #define PN532_WAKEUP_BYTE (0x55U)
- #define PN532_ACK_LEN (6U)
- #define PN532_FRAME_TFI_OVERHEAD (1U)
- #define PN532_FIRMWARE_CMD_LEN (1U)
- #define PN532_FIRMWARE_RESP_LEN (13U) /∗ matches Adafruit_PN532::getFirmwareVersion ∗/
- #define PN532_FIRMWARE_HDR_LEN (7U)
- #define PN532_FW_IC_OFFSET (7U)
- #define PN532_FW_VER_OFFSET (8U)
- #define PN532_FW_REV_OFFSET (9U)

- #define PN532_FW_SUPPORT_OFFSET (10U)
- #define PN532_SAM_CMD_LEN (4U)
- #define PN532_SAM_RESP_LEN (9U)
- #define PN532_SAM_RESP_CODE_OFFSET (6U)
- #define PN532_SAM_RESP_CODE (0x15U) /* SAMCONFIGURATION + 1 */
- #define PN532_SAM_NORMAL_MODE (0x01U)
- #define PN532_SAM_TIMEOUT (0x14U) /* 50 ms x 20 = 1 s */
- #define PN532_SAM_USE_IRQ (0x01U)
- #define PN532_PASSIVE_CMD_LEN (3U)
- #define PN532_PASSIVE_RESP_LEN (64U)
- #define PN532_PASSIVE_MAX_TARGETS (1U)
- #define PN532_PASSIVE_NUM_TARGETS_OFFSET (7U)
- #define PN532_PASSIVE_EXPECTED_TARGETS (1U)
- #define PN532_PASSIVE_UID_LEN_OFFSET (12U)
- #define PN532_PASSIVE_UID_DATA_OFFSET (13U)
- #define PN532_BYTE_SHIFT_BITS (8U)
- #define PN532_EXCHANGE_CMD_OVERHEAD (2U)
- #define PN532_EXCHANGE_TG (0x01U)
- #define PN532_EXCHANGE_FRAME_MAX (440U)
- #define PN532_EXCHANGE_LEN_OFFSET (3U)
- #define PN532_EXCHANGE_STATUS_OFFSET (7U)
- #define PN532_EXCHANGE_DATA_OFFSET (8U) /* frame[8] = first DataOut byte for normal frames */
- #define PN532_EXCHANGE_LEN_BIAS (3U) /* LEN covers D5 + CMD + ERR; DataOut = LEN - 3 */
- #define PN532_EXCHANGE_STATUS_OK (0x00U)
- #define PN532_FRAME_HDR_LEN (5U)
- #define PN532_FRAME_TAIL_LEN (2U)
- #define PN532_EXT_FRAME_INDICATOR (0xFFU)
- #define PN532_EXT_FRAME_HDR_LEN (8U)
- #define PN532_EXT_FRAME_LENHI_OFFSET (5U)
- #define PN532_EXT_FRAME_LENLO_OFFSET (6U)
- #define PN532_EXT_EXCHANGE_ERR_OFFSET (10U)
- #define PN532_EXT_EXCHANGE_DATA_OFFSET (11U)
- #define PN532_INRELEASE_CMD_LEN (2U)
- #define PN532_INRELEASE_RESP_LEN (10U)

Functions

- static bool pn532_buffer_equal (const uint8_t *lhs, const uint8_t *rhs, uint8_t len)
- static void spi_write_byte (pn532_t *dev, uint8_t data)
- static uint8_t spi_read_byte (pn532_t *dev)
- static uint8_t i2c_read_ready (pn532_t *dev)
- static uint8_t read_ready (pn532_t *dev)
- static void read_data (pn532_t *dev, uint8_t *buff, uint8_t n)
- static bool check_ack (pn532_t *dev)
- static void write_command (pn532_t *dev, const uint8_t *cmd, uint8_t cmd_len)
- static bool send_command_check_ack (pn532_t *dev, const uint8_t *cmd, uint8_t cmd_len, uint16_t timeout)
- static uint16_t read_data_apdu_frame (pn532_t *dev, uint8_t *buff, uint16_t max_len)
- static esp_err_t pn532_init_spi (pn532_t *dev, const pn532_config_t *config)
- static esp_err_t pn532_init_i2c (pn532_t *dev, const pn532_config_t *config)
- esp_err_t pn532_init (pn532_t *dev, const pn532_config_t *config)

  Initialise the PN532 and bring it to a ready state.
- uint32_t pn532_get_firmware_version (pn532_t *dev)

Query the PN532 firmware version.

- bool pn532_sam_config (pn532_t *dev)

  Configure the PN532's Security Access Module (SAM).

- uint32_t pn532_read_passive_target_id (pn532_t *dev, uint8_t cardbaudrate)

  Scan for a passive ISO 14443-A card and return its UID.

- bool pn532_send_apdu (pn532_t *dev, const uint8_t *apdu, uint8_t apdu_len, uint8_t *response, uint16_t *response_len)

  Exchange a single ISO-DEP APDU with the currently selected card.

- bool pn532_release_target (pn532_t *dev)

  Release the currently selected NFC target.

Variables

- static const char *const PN532_LOG_TAG = "pn532"
- static const uint8_t pn532_ack [PN532_ACK_LEN]
- static const uint8_t pn532_response_fw [PN532_FIRMWARE_HDR_LEN]

5.19.1 Macro Definition Documentation

5.19.1.1 GPIO_LEVEL_HIGH

```
#define GPIO_LEVEL_HIGH (1U) /* drive pin high */
```

Definition at line 90 of file pn532.c.
Referenced by pn532_init_i2c(), pn532_init_spi(), read_data(), read_data_apdu_frame(), read_ready(), and write_command().

5.19.1.2 GPIO_LEVEL_LOW

```
#define GPIO_LEVEL_LOW (0U) /* drive pin low */
```

Definition at line 89 of file pn532.c.
Referenced by pn532_init_i2c(), pn532_init_spi(), read_data(), read_data_apdu_frame(), read_ready(), and write_command().

5.19.1.3 GPIO_PIN_BITMASK_BASE

```
#define GPIO_PIN_BITMASK_BASE (1ULL) /* base bit for gpio_config_t.pin_bit_mask */
```

Definition at line 91 of file pn532.c.
Referenced by pn532_init_i2c(), and pn532_init_spi().

5.19.1.4 PN532_ACK_LEN

```
#define PN532_ACK_LEN (6U)
```

Definition at line 103 of file pn532.c.
Referenced by check_ack().

5.19.1.5 PN532_APDU_TIMEOUT_MS

```
#define PN532_APDU_TIMEOUT_MS (5000U)
```

Definition at line 82 of file pn532.c.
Referenced by pn532_send_apdu().

5.19.1.6 PN532_BYTE_DELAY_MS

```
#define PN532_BYTE_DELAY_MS (1U)
```

Definition at line 83 of file pn532.c.
Referenced by read_data(), and read_data_apdu_frame().

5.19.1.7 PN532_BYTE_SHIFT_BITS

```
#define PN532_BYTE_SHIFT_BITS (8U)
```
Definition at line 147 of file pn532.c.
Referenced by pn532_get_firmware_version(), and pn532_read_passive_target_id().

5.19.1.8 PN532_CMD_TIMEOUT_MS

```
#define PN532_CMD_TIMEOUT_MS (1000U)
```
Definition at line 80 of file pn532.c.
Referenced by pn532_get_firmware_version(), pn532_read_passive_target_id(), pn532_release_target(), and pn532_sam_config().

5.19.1.9 PN532_CS_TOGGLE_DELAY_MS

```
#define PN532_CS_TOGGLE_DELAY_MS (2U)
```
Definition at line 76 of file pn532.c.
Referenced by pn532_init_spi(), read_data(), read_data_apdu_frame(), read_ready(), and write_command().

5.19.1.10 PN532_EXCHANGE_CMD_OVERHEAD

```
#define PN532_EXCHANGE_CMD_OVERHEAD (2U)
```
Definition at line 153 of file pn532.c.
Referenced by pn532_send_apdu().

5.19.1.11 PN532_EXCHANGE_DATA_OFFSET

```
#define PN532_EXCHANGE_DATA_OFFSET (8U) /* frame[8] = first DataOut byte for normal frames */
```
Definition at line 159 of file pn532.c.
Referenced by pn532_send_apdu().

5.19.1.12 PN532_EXCHANGE_FRAME_MAX

```
#define PN532_EXCHANGE_FRAME_MAX (440U)
```
Definition at line 156 of file pn532.c.
Referenced by pn532_send_apdu(), and read_data_apdu_frame().

5.19.1.13 PN532_EXCHANGE_LEN_BIAS

```
#define PN532_EXCHANGE_LEN_BIAS (3U) /* LEN covers D5 + CMD + ERR; DataOut = LEN - 3 */
```
Definition at line 160 of file pn532.c.
Referenced by pn532_send_apdu().

5.19.1.14 PN532_EXCHANGE_LEN_OFFSET

```
#define PN532_EXCHANGE_LEN_OFFSET (3U)
```
Definition at line 157 of file pn532.c.
Referenced by pn532_send_apdu(), and read_data_apdu_frame().

5.19.1.15 PN532_EXCHANGE_STATUS_OFFSET

```
#define PN532_EXCHANGE_STATUS_OFFSET (7U)
```
Definition at line 158 of file pn532.c.
Referenced by pn532_release_target(), and pn532_send_apdu().

5.19.1.16 PN532_EXCHANGE_STATUS_OK

```
#define PN532_EXCHANGE_STATUS_OK (0x00U)
```
Definition at line 161 of file pn532.c.
Referenced by pn532_release_target(), and pn532_send_apdu().

5.19.1.17 PN532_EXCHANGE_TG

```
#define PN532_EXCHANGE_TG (0x01U)
```
Definition at line 154 of file pn532.c.
Referenced by pn532_release_target(), and pn532_send_apdu().

5.19.1.18 PN532_EXT_EXCHANGE_DATA_OFFSET

```
#define PN532_EXT_EXCHANGE_DATA_OFFSET (11U)
```
Definition at line 177 of file pn532.c.
Referenced by pn532_send_apdu().

5.19.1.19 PN532_EXT_EXCHANGE_ERR_OFFSET

```
#define PN532_EXT_EXCHANGE_ERR_OFFSET (10U)
```
Definition at line 176 of file pn532.c.
Referenced by pn532_send_apdu().

5.19.1.20 PN532_EXT_FRAME_HDR_LEN

```
#define PN532_EXT_FRAME_HDR_LEN (8U)
```
Definition at line 173 of file pn532.c.
Referenced by read_data_apdu_frame().

5.19.1.21 PN532_EXT_FRAME_INDICATOR

```
#define PN532_EXT_FRAME_INDICATOR (0xFFU)
```
Definition at line 172 of file pn532.c.
Referenced by pn532_send_apdu(), and read_data_apdu_frame().

5.19.1.22 PN532_EXT_FRAME_LENHI_OFFSET

```
#define PN532_EXT_FRAME_LENHI_OFFSET (5U)
```
Definition at line 174 of file pn532.c.
Referenced by pn532_send_apdu(), and read_data_apdu_frame().

5.19.1.23 PN532_EXT_FRAME_LENLO_OFFSET

```
#define PN532_EXT_FRAME_LENLO_OFFSET (6U)
```
Definition at line 175 of file pn532.c.
Referenced by pn532_send_apdu(), and read_data_apdu_frame().

5.19.1.24 PN532_FIRMWARE_CMD_LEN

```
#define PN532_FIRMWARE_CMD_LEN (1U)
```
Definition at line 116 of file pn532.c.
Referenced by pn532_get_firmware_version().

5.19.1.25 PN532_FIRMWARE_HDR_LEN

```
#define PN532_FIRMWARE_HDR_LEN (7U)
```
Definition at line 118 of file pn532.c.
Referenced by pn532_get_firmware_version().

5.19.1.26 PN532_FIRMWARE_RESP_LEN

```
#define PN532_FIRMWARE_RESP_LEN (13U) /* matches Adafruit_PN532::getFirmwareVersion */
```

Definition at line 117 of file pn532.c.
Referenced by pn532_get_firmware_version().

5.19.1.27 PN532_FIRMWAREVERSION

```
#define PN532_FIRMWAREVERSION (0x02U)
```

Definition at line 42 of file pn532.c.
Referenced by pn532_get_firmware_version().

5.19.1.28 PN532_FRAME_HDR_LEN

```
#define PN532_FRAME_HDR_LEN (5U)
```

Definition at line 165 of file pn532.c.
Referenced by read_data_apdu_frame().

5.19.1.29 PN532_FRAME_TAIL_LEN

```
#define PN532_FRAME_TAIL_LEN (2U)
```

Definition at line 166 of file pn532.c.
Referenced by read_data_apdu_frame().

5.19.1.30 PN532_FRAME_TFI_OVERHEAD

```
#define PN532_FRAME_TFI_OVERHEAD (1U)
```

Definition at line 110 of file pn532.c.
Referenced by write_command().

5.19.1.31 PN532_FW_IC_OFFSET

```
#define PN532_FW_IC_OFFSET (7U)
```

Definition at line 119 of file pn532.c.
Referenced by pn532_get_firmware_version().

5.19.1.32 PN532_FW_REV_OFFSET

```
#define PN532_FW_REV_OFFSET (9U)
```

Definition at line 121 of file pn532.c.
Referenced by pn532_get_firmware_version().

5.19.1.33 PN532_FW_SUPPORT_OFFSET

```
#define PN532_FW_SUPPORT_OFFSET (10U)
```

Definition at line 122 of file pn532.c.
Referenced by pn532_get_firmware_version().

5.19.1.34 PN532_FW_VER_OFFSET

```
#define PN532_FW_VER_OFFSET (8U)
```

Definition at line 120 of file pn532.c.
Referenced by pn532_get_firmware_version().

5.19.1.35 PN532_HOSTTOPN532

```
#define PN532_HOSTTOPN532 (0xD4U)
```

Definition at line 36 of file pn532.c.
Referenced by write_command().

5.19.1.36 PN532_I2C_RX_MAX

```
#define PN532_I2C_RX_MAX (200U)
```
Definition at line 70 of file pn532.c.
Referenced by read_data().

5.19.1.37 PN532_I2C_TIMEOUT_MS

```
#define PN532_I2C_TIMEOUT_MS (100)
```
Definition at line 66 of file pn532.c.
Referenced by i2c_read_ready(), read_data(), read_data_apdu_frame(), and write_command().

5.19.1.38 PN532_I2C_TX_MAX

```
#define PN532_I2C_TX_MAX (PN532_MAX_APDU_LEN + 16U)
```
Definition at line 68 of file pn532.c.
Referenced by write_command().

5.19.1.39 PN532_INDATAEXCHANGE

```
#define PN532_INDATAEXCHANGE (0x40U)
```
Definition at line 45 of file pn532.c.
Referenced by pn532_send_apdu().

5.19.1.40 PN532_INLISTPASSIVETARGET

```
#define PN532_INLISTPASSIVETARGET (0x4AU)
```
Definition at line 44 of file pn532.c.
Referenced by pn532_read_passive_target_id().

5.19.1.41 PN532_INRELEASE

```
#define PN532_INRELEASE (0x52U)
```
Definition at line 46 of file pn532.c.
Referenced by pn532_release_target().

5.19.1.42 PN532_INRELEASE_CMD_LEN

```
#define PN532_INRELEASE_CMD_LEN (2U)
```
Definition at line 184 of file pn532.c.
Referenced by pn532_release_target().

5.19.1.43 PN532_INRELEASE_RESP_LEN

```
#define PN532_INRELEASE_RESP_LEN (10U)
```
Definition at line 185 of file pn532.c.
Referenced by pn532_release_target().

5.19.1.44 PN532_PASSIVE_CMD_LEN

```
#define PN532_PASSIVE_CMD_LEN (3U)
```
Definition at line 140 of file pn532.c.
Referenced by pn532_read_passive_target_id().

5.19.1.45 PN532_PASSIVE_EXPECTED_TARGETS

```
#define PN532_PASSIVE_EXPECTED_TARGETS (1U)
```
Definition at line 144 of file pn532.c.
Referenced by pn532_read_passive_target_id().

5.19.1.46 PN532_PASSIVE_MAX_TARGETS

```
#define PN532_PASSIVE_MAX_TARGETS (1U)
```
Definition at line 142 of file pn532.c.
Referenced by pn532_read_passive_target_id().

5.19.1.47 PN532_PASSIVE_NUM_TARGETS_OFFSET

```
#define PN532_PASSIVE_NUM_TARGETS_OFFSET (7U)
```
Definition at line 143 of file pn532.c.
Referenced by pn532_read_passive_target_id().

5.19.1.48 PN532_PASSIVE_RESP_LEN

```
#define PN532_PASSIVE_RESP_LEN (64U)
```
Definition at line 141 of file pn532.c.
Referenced by pn532_read_passive_target_id().

5.19.1.49 PN532_PASSIVE_UID_DATA_OFFSET

```
#define PN532_PASSIVE_UID_DATA_OFFSET (13U)
```
Definition at line 146 of file pn532.c.
Referenced by pn532_read_passive_target_id().

5.19.1.50 PN532_PASSIVE_UID_LEN_OFFSET

```
#define PN532_PASSIVE_UID_LEN_OFFSET (12U)
```
Definition at line 145 of file pn532.c.
Referenced by pn532_read_passive_target_id().

5.19.1.51 PN532_POLL_INTERVAL_MS

```
#define PN532_POLL_INTERVAL_MS (10U)
```
Definition at line 79 of file pn532.c.
Referenced by send_command_check_ack().

5.19.1.52 PN532_POSTAMBLE

```
#define PN532_POSTAMBLE (0x00U)
```
Definition at line 35 of file pn532.c.
Referenced by write_command().

5.19.1.53 PN532_PREAMBLE

```
#define PN532_PREAMBLE (0x00U)
```
Definition at line 31 of file pn532.c.
Referenced by pn532_init_spi(), and write_command().

5.19.1.54 PN532_SAM_CMD_LEN

```
#define PN532_SAM_CMD_LEN (4U)
```
Definition at line 128 of file pn532.c.
Referenced by pn532_sam_config().

5.19.1.55 PN532_SAM_NORMAL_MODE

```
#define PN532_SAM_NORMAL_MODE (0x01U)
```
Definition at line 132 of file pn532.c.
Referenced by pn532_sam_config().

5.19.1.56 PN532_SAM_RESP_CODE

```
#define PN532_SAM_RESP_CODE (0x15U) /* SAMCONFIGURATION + 1 */
```
Definition at line 131 of file pn532.c.
Referenced by pn532_sam_config().

5.19.1.57 PN532_SAM_RESP_CODE_OFFSET

```
#define PN532_SAM_RESP_CODE_OFFSET (6U)
```
Definition at line 130 of file pn532.c.
Referenced by pn532_sam_config().

5.19.1.58 PN532_SAM_RESP_LEN

```
#define PN532_SAM_RESP_LEN (9U)
```
Definition at line 129 of file pn532.c.
Referenced by pn532_sam_config().

5.19.1.59 PN532_SAM_TIMEOUT

```
#define PN532_SAM_TIMEOUT (0x14U) /* 50 ms x 20 = 1 s */
```
Definition at line 133 of file pn532.c.
Referenced by pn532_sam_config().

5.19.1.60 PN532_SAM_USE_IRQ

```
#define PN532_SAM_USE_IRQ (0x01U)
```
Definition at line 134 of file pn532.c.
Referenced by pn532_sam_config().

5.19.1.61 PN532_SAMCONFIGURATION

```
#define PN532_SAMCONFIGURATION (0x14U)
```
Definition at line 43 of file pn532.c.
Referenced by pn532_sam_config().

5.19.1.62 PN532_SPI_BITS

```
#define PN532_SPI_BITS (8U)
```
Definition at line 60 of file pn532.c.
Referenced by spi_read_byte(), and spi_write_byte().

5.19.1.63 PN532_SPI_CLOCK_HZ

```
#define PN532_SPI_CLOCK_HZ (1000000U)
```
Definition at line 56 of file pn532.c.
Referenced by pn532_init_spi().

5.19.1.64 PN532_SPI_DATAREAD

```
#define PN532_SPI_DATAREAD (0x03U)
```
Definition at line 54 of file pn532.c.
Referenced by read_data(), and read_data_apdu_frame().

5.19.1.65 PN532_SPI_DATAWRITE

```
#define PN532_SPI_DATAWRITE (0x01U)
```
Definition at line 53 of file pn532.c.
Referenced by write_command().

5.19.1.66 PN532_SPI_MODE

```
#define PN532_SPI_MODE (0U)
```
Definition at line 57 of file pn532.c.
Referenced by pn532_init_spi().

5.19.1.67 PN532_SPI_NO_PIN

```
#define PN532_SPI_NO_PIN (-1)
```
Definition at line 58 of file pn532.c.
Referenced by pn532_init_spi().

5.19.1.68 PN532_SPI_QUEUE_SIZE

```
#define PN532_SPI_QUEUE_SIZE (1U)
```
Definition at line 59 of file pn532.c.
Referenced by pn532_init_spi().

5.19.1.69 PN532_SPI_READY

```
#define PN532_SPI_READY (0x01U)
```
Definition at line 55 of file pn532.c.
Referenced by send_command_check_ack().

5.19.1.70 PN532_SPI_STATREAD

```
#define PN532_SPI_STATREAD (0x02U)
```
Definition at line 52 of file pn532.c.
Referenced by read_ready().

5.19.1.71 PN532_STARTCODE1

```
#define PN532_STARTCODE1 (0x00U)
```
Definition at line 33 of file pn532.c.

5.19.1.72 PN532_STARTCODE2

```
#define PN532_STARTCODE2 (0xFFU)
```
Definition at line 34 of file pn532.c.
Referenced by write_command().

5.19.1.73 PN532_SYNC_DELAY_MS

```
#define PN532_SYNC_DELAY_MS (100U)
```
Definition at line 78 of file pn532.c.
Referenced by pn532_init().

5.19.1.74 PN532_WAKEUP_BYTE

```
#define PN532_WAKEUP_BYTE (0x55U)
```
Definition at line 97 of file pn532.c.
Referenced by pn532_init_spi().

5.19.1.75 PN532_WAKEUP_DELAY_MS

```
#define PN532_WAKEUP_DELAY_MS (1000U)
```
Definition at line 77 of file pn532.c.
Referenced by pn532_init_spi().

5.19.2 Function Documentation

5.19.2.1 check_ack()

```
bool check_ack (
            pn532_t * dev)  [static]
```

Definition at line 291 of file pn532.c.
References pn532_ack, PN532_ACK_LEN, pn532_buffer_equal(), and read_data().
Referenced by send_command_check_ack().

5.19.2.2 i2c_read_ready()

```
uint8_t i2c_read_ready (
            pn532_t * dev)  [static]
```

Definition at line 238 of file pn532.c.
References pn532_t::i2c_dev, and PN532_I2C_TIMEOUT_MS.
Referenced by read_ready().

5.19.2.3 pn532_buffer_equal()

```
bool pn532_buffer_equal (
            const uint8_t * lhs,
            const uint8_t * rhs,
            uint8_t len)  [static]
```

Definition at line 301 of file pn532.c.
Referenced by check_ack(), and pn532_get_firmware_version().

5.19.2.4 pn532_init_i2c()

```
esp_err_t pn532_init_i2c (
            pn532_t * dev,
            const pn532_config_t * config)  [static]
```

Definition at line 579 of file pn532.c.
References GPIO_LEVEL_HIGH, GPIO_LEVEL_LOW, GPIO_PIN_BITMASK_BASE, pn532_t::i2c_bus, pn532_config_t::i2c_clock_hz, pn532_t::i2c_dev, pn532_config_t::i2c_port, pn532_config_t::pin_irq, pn532_t::pin_irq, pn532_config_t::pin_rst, pn532_t::pin_rst, pn532_config_t::pin_scl, pn532_config_t::pin_sda, PN532_I2C_ADDRESS, and PN532_LOG_TAG.
Referenced by pn532_init().

5.19.2.5 pn532_init_spi()

```
esp_err_t pn532_init_spi (
            pn532_t * dev,
            const pn532_config_t * config)  [static]
```

Definition at line 519 of file pn532.c.
References GPIO_LEVEL_HIGH, GPIO_LEVEL_LOW, GPIO_PIN_BITMASK_BASE, pn532_config_t::pin_cs, pn532_t::pin_cs, pn532_config_t::pin_miso, pn532_config_t::pin_mosi, pn532_config_t::pin_sclk, PN532_CS_TOGGLE_DELAY_MS, PN532_PREAMBLE, PN532_SPI_CLOCK_HZ, PN532_SPI_MODE, PN532_SPI_NO_PIN, PN532_SPI_QUEUE_SIZE, PN532_WAKEUP_BYTE, PN532_WAKEUP_DELAY_MS, pn532_config_t::skip_bus_init, pn532_t::spi, pn532_config_t::spi_host, and spi_write_byte().
Referenced by pn532_init().

5.19.2.6 read_data()

```
void read_data (
            pn532_t * dev,
            uint8_t * buff,
            uint8_t n)  [static]
```

Definition at line 266 of file pn532.c.
References GPIO_LEVEL_HIGH, GPIO_LEVEL_LOW, pn532_t::i2c_dev, pn532_t::pin_cs, PN532_BYTE_DELAY_MS, PN532_CS_TOGGLE_DELAY_MS, PN532_I2C_RX_MAX,

PN532_I2C_TIMEOUT_MS, PN532_SPI_DATAREAD, PN532_TRANSPORT_I2C, spi_read_byte(), spi_write_byte(), and pn532_t::transport.
Referenced by check_ack(), pn532_get_firmware_version(), pn532_read_passive_target_id(), pn532_release_target(), and pn532_sam_config().

5.19.2.7 read_data_apdu_frame()

```
uint16_t read_data_apdu_frame (
            pn532_t * dev,
            uint8_t * buff,
            uint16_t max_len)  [static]
```

Definition at line 428 of file pn532.c.
References GPIO_LEVEL_HIGH, GPIO_LEVEL_LOW, pn532_t::i2c_dev, pn532_t::pin_cs, PN532_BYTE_DELAY_MS, PN532_CS_TOGGLE_DELAY_MS, PN532_EXCHANGE_FRAME_MAX, PN532_EXCHANGE_LEN_OFFSET, PN532_EXT_FRAME_HDR_LEN, PN532_EXT_FRAME_INDICATOR, PN532_EXT_FRAME_LENHI_OFFSET, PN532_EXT_FRAME_LENLO_OFFSET, PN532_FRAME_HDR_LEN, PN532_FRAME_TAIL_LEN, PN532_I2C_TIMEOUT_MS, PN532_SPI_DATAREAD, PN532_TRANSPORT_I2C, spi_read_byte(), spi_write_byte(), and pn532_t::transport.
Referenced by pn532_send_apdu().

5.19.2.8 read_ready()

```
uint8_t read_ready (
            pn532_t * dev)  [static]
```

Definition at line 249 of file pn532.c.
References GPIO_LEVEL_HIGH, GPIO_LEVEL_LOW, i2c_read_ready(), pn532_t::pin_cs, PN532_CS_TOGGLE_DELAY_MS, PN532_SPI_STATREAD, PN532_TRANSPORT_I2C, spi_read_byte(), spi_write_byte(), and pn532_t::transport.
Referenced by send_command_check_ack().

5.19.2.9 send_command_check_ack()

```
bool send_command_check_ack (
            pn532_t * dev,
            const uint8_t * cmd,
            uint8_t cmd_len,
            uint16_t timeout)  [static]
```

Definition at line 367 of file pn532.c.
References check_ack(), PN532_LOG_TAG, PN532_POLL_INTERVAL_MS, PN532_SPI_READY, read_ready(), and write_command().
Referenced by pn532_get_firmware_version(), pn532_read_passive_target_id(), pn532_release_target(), pn532_sam_config(), and pn532_send_apdu().

5.19.2.10 spi_read_byte()

```
uint8_t spi_read_byte (
            pn532_t * dev)  [static]
```

Definition at line 220 of file pn532.c.
References PN532_SPI_BITS, and pn532_t::spi.
Referenced by read_data(), read_data_apdu_frame(), and read_ready().

5.19.2.11 spi_write_byte()

```
void spi_write_byte (
            pn532_t * dev,
            uint8_t data)  [static]
```

Definition at line 211 of file pn532.c.
References PN532_SPI_BITS, and pn532_t::spi.
Referenced by pn532_init_spi(), read_data(), read_data_apdu_frame(), read_ready(), and write_command().

5.19.2.12 write_command()

```
void write_command (
            pn532_t * dev,
            const uint8_t * cmd,
            uint8_t cmd_len)  [static]
```

Definition at line 315 of file pn532.c.
References GPIO_LEVEL_HIGH, GPIO_LEVEL_LOW, pn532_t::i2c_dev, pn532_t::pin_cs, PN532_CS_TOGGLE_DELAY_MS, PN532_FRAME_TFI_OVERHEAD, PN532_HOSTTOPN532, PN532_I2C_TIMEOUT_MS, PN532_I2C_TX_MAX, PN532_POSTAMBLE, PN532_PREAMBLE, PN532_SPI_DATAWRITE, PN532_STARTCODE2, PN532_TRANSPORT_I2C, spi_write_byte(), and pn532_t::transport.
Referenced by send_command_check_ack().

5.19.3 Variable Documentation

5.19.3.1 pn532_ack

```
const uint8_t pn532_ack[PN532_ACK_LEN]  [static]
```

Initial value:

```
= {
    0x00U, 0x00U, 0xFFU, 0x00U, 0xFFU, 0x00U
}
```

Definition at line 192 of file pn532.c.
Referenced by check_ack().

5.19.3.2 PN532_LOG_TAG

```
const char* const PN532_LOG_TAG = "pn532"  [static]
```

Definition at line 25 of file pn532.c.
Referenced by pn532_get_firmware_version(), pn532_init(), pn532_init_i2c(), pn532_send_apdu(), and send_command_check_ack().

5.19.3.3 pn532_response_fw

```
const uint8_t pn532_response_fw[PN532_FIRMWARE_HDR_LEN]  [static]
```

Initial value:

```
= {
    0x00U, 0x00U, 0xFFU, 0x06U, 0xFAU, 0xD5U, 0x03U
}
```

Definition at line 197 of file pn532.c.
Referenced by pn532_get_firmware_version().

5.20 pn532.c

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later AND BSD-3-Clause
00003  *
00004  * PN532 driver for ESP-IDF -- supports SPI and I²C transports.
00005  *
00006  * Portions of this file derive from the Adafruit_PN532 Arduino library
00007  * (Copyright (c) 2012, Adafruit Industries; BSD-3-Clause). The full
00008  * upstream copyright notice and license text are reproduced in
00009  * NOTICES.md at the repository root.
00010  *
00011  * Additions on top of the upstream driver -- the I²C transport via the
00012  * IDF v5.x i2c_master API and the PN532 extended-frame (8-byte header)
00013  * parser -- are licensed under LGPL-3.0-or-later (with a commercial
00014  * option, see LICENSE / COMMERCIAL.md).
00015  */
00016
00017 #include "pn532.h"
00018 #include "driver/gpio.h"
00019 #include "driver/i2c_master.h"   /* new IDF v5.x master API */
00020 #include "esp_log.h"
```

```
00021 #include "freertos/FreeRTOS.h"
00022 #include "freertos/task.h"
00023 #include <string.h>
00024
00025 static const char *const PN532_LOG_TAG = "pn532";
00026
00027 /**************************************************************************
00028  * PN532 frame constants
00029  **************************************************************************/
00030
00031 #define PN532_PREAMBLE      (0x00U)
00032 /* cppcheck-suppress misra-c2012-2.5 */
00033 #define PN532_STARTCODE1    (0x00U)
00034 #define PN532_STARTCODE2    (0xFFU)
00035 #define PN532_POSTAMBLE     (0x00U)
00036 #define PN532_HOSTTOPN532   (0xD4U)
00037
00038 /**************************************************************************
00039  * PN532 command codes
00040  **************************************************************************/
00041
00042 #define PN532_FIRMWAREVERSION       (0x02U)
00043 #define PN532_SAMCONFIGURATION      (0x14U)
00044 #define PN532_INLISTPASSIVETARGET   (0x4AU)
00045 #define PN532_INDATAEXCHANGE        (0x40U)
00046 #define PN532_INRELEASE             (0x52U)
00047
00048 /**************************************************************************
00049  * SPI protocol constants
00050  **************************************************************************/
00051
00052 #define PN532_SPI_STATREAD     (0x02U)
00053 #define PN532_SPI_DATAWRITE    (0x01U)
00054 #define PN532_SPI_DATAREAD     (0x03U)
00055 #define PN532_SPI_READY        (0x01U)
00056 #define PN532_SPI_CLOCK_HZ     (1000000U)
00057 #define PN532_SPI_MODE         (0U)
00058 #define PN532_SPI_NO_PIN       (-1)
00059 #define PN532_SPI_QUEUE_SIZE   (1U)
00060 #define PN532_SPI_BITS         (8U)
00061
00062 /**************************************************************************
00063  * I2C protocol constants
00064  **************************************************************************/
00065
00066 #define PN532_I2C_TIMEOUT_MS   (100)
00067 /* Max frame bytes the host may send to PN532 (preamble..postamble). */
00068 #define PN532_I2C_TX_MAX       (PN532_MAX_APDU_LEN + 16U)
00069 /* Max read length used anywhere (PASSIVE response is the largest at 64). */
00070 #define PN532_I2C_RX_MAX       (200U)
00071
00072 /**************************************************************************
00073  * Timing constants (milliseconds)
00074  **************************************************************************/
00075
00076 #define PN532_CS_TOGGLE_DELAY_MS   (2U)
00077 #define PN532_WAKEUP_DELAY_MS      (1000U)
00078 #define PN532_SYNC_DELAY_MS        (100U)
00079 #define PN532_POLL_INTERVAL_MS     (10U)
00080 #define PN532_CMD_TIMEOUT_MS       (1000U)
00081 /* Card ECDSA in getCardCertificate takes up to 3 s; use 5 s margin. */
00082 #define PN532_APDU_TIMEOUT_MS      (5000U)
00083 #define PN532_BYTE_DELAY_MS        (1U)
00084
00085 /**************************************************************************
00086  * GPIO level constants
00087  **************************************************************************/
00088
00089 #define GPIO_LEVEL_LOW         (0U)    /* drive pin low  */
00090 #define GPIO_LEVEL_HIGH        (1U)    /* drive pin high */
00091 #define GPIO_PIN_BITMASK_BASE (1ULL)  /* base bit for gpio_config_t.pin_bit_mask */
00092
00093 /**************************************************************************
00094  * Wakeup sequence constants
00095  **************************************************************************/
00096
00097 #define PN532_WAKEUP_BYTE   (0x55U)
00098
00099 /**************************************************************************
00100  * ACK frame
00101  **************************************************************************/
00102
00103 #define PN532_ACK_LEN   (6U)
00104
00105 /**************************************************************************
00106  * Frame structure constants
00107  **************************************************************************/
```

```
00108
00109 /* HOSTTOPN532 TFI byte added to cmd_len when building a frame. */
00110 #define PN532_FRAME_TFI_OVERHEAD  (1U)
00111
00112 /**************************************************************************
00113  * Firmware version response offsets and lengths
00114  **************************************************************************/
00115
00116 #define PN532_FIRMWARE_CMD_LEN     (1U)
00117 #define PN532_FIRMWARE_RESP_LEN    (13U)  /* matches Adafruit_PN532::getFirmwareVersion */
00118 #define PN532_FIRMWARE_HDR_LEN     (7U)
00119 #define PN532_FW_IC_OFFSET         (7U)
00120 #define PN532_FW_VER_OFFSET        (8U)
00121 #define PN532_FW_REV_OFFSET        (9U)
00122 #define PN532_FW_SUPPORT_OFFSET    (10U)
00123
00124 /**************************************************************************
00125  * SAMConfiguration constants
00126  **************************************************************************/
00127
00128 #define PN532_SAM_CMD_LEN           (4U)
00129 #define PN532_SAM_RESP_LEN          (9U)
00130 #define PN532_SAM_RESP_CODE_OFFSET  (6U)
00131 #define PN532_SAM_RESP_CODE         (0x15U)  /* SAMCONFIGURATION + 1 */
00132 #define PN532_SAM_NORMAL_MODE       (0x01U)
00133 #define PN532_SAM_TIMEOUT           (0x14U)  /* 50 ms x 20 = 1 s */
00134 #define PN532_SAM_USE_IRQ           (0x01U)
00135
00136 /**************************************************************************
00137  * InListPassiveTarget constants
00138  **************************************************************************/
00139
00140 #define PN532_PASSIVE_CMD_LEN             (3U)
00141 #define PN532_PASSIVE_RESP_LEN            (64U)
00142 #define PN532_PASSIVE_MAX_TARGETS         (1U)
00143 #define PN532_PASSIVE_NUM_TARGETS_OFFSET  (7U)
00144 #define PN532_PASSIVE_EXPECTED_TARGETS    (1U)
00145 #define PN532_PASSIVE_UID_LEN_OFFSET      (12U)
00146 #define PN532_PASSIVE_UID_DATA_OFFSET     (13U)
00147 #define PN532_BYTE_SHIFT_BITS             (8U)
00148
00149 /**************************************************************************
00150  * InDataExchange constants
00151  **************************************************************************/
00152
00153 #define PN532_EXCHANGE_CMD_OVERHEAD   (2U)
00154 #define PN532_EXCHANGE_TG             (0x01U)
00155 /* Extended frames: up to 8-byte header + 418-byte body (415-byte DataOut) + 2-byte tail = 428; use
      440 for margin. */
00156 #define PN532_EXCHANGE_FRAME_MAX      (440U)
00157 #define PN532_EXCHANGE_LEN_OFFSET     (3U)
00158 #define PN532_EXCHANGE_STATUS_OFFSET  (7U)
00159 #define PN532_EXCHANGE_DATA_OFFSET    (8U)    /* frame[8] = first DataOut byte for normal frames */
00160 #define PN532_EXCHANGE_LEN_BIAS       (3U)    /* LEN covers D5 + CMD + ERR; DataOut = LEN - 3 */
00161 #define PN532_EXCHANGE_STATUS_OK      (0x00U)
00162
00163 /* PN532 normal-frame structure: 5-byte header (preamble+start1+start2+LEN+LCS)
00164  * followed by LEN data bytes, then 2-byte tail (DCS+postamble). */
00165 #define PN532_FRAME_HDR_LEN   (5U)
00166 #define PN532_FRAME_TAIL_LEN  (2U)
00167
00168 /* PN532 extended-frame indicator and structure.
00169  * Triggered when frame[3]==0xFF and frame[4]==0xFF.
00170  * Header is 8 bytes: preamble+start1+start2+FF+FF+LEN_H+LEN_L+LCS.
00171  * ERR is at offset 10; DataOut starts at offset 11. */
00172 #define PN532_EXT_FRAME_INDICATOR      (0xFFU)
00173 #define PN532_EXT_FRAME_HDR_LEN        (8U)
00174 #define PN532_EXT_FRAME_LENHI_OFFSET   (5U)
00175 #define PN532_EXT_FRAME_LENLO_OFFSET   (6U)
00176 #define PN532_EXT_EXCHANGE_ERR_OFFSET  (10U)
00177 #define PN532_EXT_EXCHANGE_DATA_OFFSET (11U)
00178
00179
00180 /**************************************************************************
00181  * InRelease constants
00182  **************************************************************************/
00183
00184 #define PN532_INRELEASE_CMD_LEN   (2U)
00185 #define PN532_INRELEASE_RESP_LEN  (10U)
00186
00187 /**************************************************************************
00188  * Module-level static data
00189  **************************************************************************/
00190
00191 /* cppcheck-suppress misra-c2012-8.9 */
00192 static const uint8_t pn532_ack[PN532_ACK_LEN] = {
00193     0x00U, 0x00U, 0xFFU, 0x00U, 0xFFU, 0x00U
```

```
00194 };
00195
00196 /* cppcheck-suppress misra-c2012-8.9 */
00197 static const uint8_t pn532_response_fw[PN532_FIRMWARE_HDR_LEN] = {
00198     0x00U, 0x00U, 0xFFU, 0x06U, 0xFAU, 0xD5U, 0x03U
00199 };
00200
00201 /**************************************************************************
00202  * Forward declarations
00203  **************************************************************************/
00204
00205 static bool pn532_buffer_equal(const uint8_t *lhs, const uint8_t *rhs, uint8_t len);
00206
00207 /**************************************************************************
00208  * Low-level SPI helpers
00209  **************************************************************************/
00210
00211 static void spi_write_byte(pn532_t *dev, uint8_t data)
00212 {
00213     spi_transaction_t t;
00214     (void)memset(&t, 0, sizeof(t));
00215     t.length = PN532_SPI_BITS;
00216     t.tx_buffer = &data;
00217     (void)spi_device_transmit(dev->spi, &t);
00218 }
00219
00220 static uint8_t spi_read_byte(pn532_t *dev)
00221 {
00222     uint8_t rx = 0U;
00223     uint8_t tx = 0x00U;
00224     spi_transaction_t t;
00225     (void)memset(&t, 0, sizeof(t));
00226     t.length = PN532_SPI_BITS;
00227     t.rxlength = PN532_SPI_BITS;
00228     t.tx_buffer = &tx;
00229     t.rx_buffer = &rx;
00230     (void)spi_device_transmit(dev->spi, &t);
00231     return rx;
00232 }
00233
00234 /**************************************************************************
00235  * Low-level I2C helpers
00236  **************************************************************************/
00237
00238 static uint8_t i2c_read_ready(pn532_t *dev)
00239 {
00240     uint8_t rdy = 0U;
00241     (void)i2c_master_receive(dev->i2c_dev, &rdy, 1U, PN532_I2C_TIMEOUT_MS);
00242     return rdy;
00243 }
00244
00245 /**************************************************************************
00246  * Transport-agnostic ready / read / write
00247  **************************************************************************/
00248
00249 static uint8_t read_ready(pn532_t *dev)
00250 {
00251     uint8_t status;
00252
00253     if (dev->transport == PN532_TRANSPORT_I2C) {
00254         status = i2c_read_ready(dev);
00255     } else {
00256         (void)gpio_set_level(dev->pin_cs, GPIO_LEVEL_LOW);
00257         vTaskDelay(pdMS_TO_TICKS(PN532_CS_TOGGLE_DELAY_MS));
00258         spi_write_byte(dev, PN532_SPI_STATREAD);
00259         status = spi_read_byte(dev);
00260         (void)gpio_set_level(dev->pin_cs, GPIO_LEVEL_HIGH);
00261     }
00262
00263     return status;
00264 }
00265
00266 static void read_data(pn532_t *dev, uint8_t *buff, uint8_t n)
00267 {
00268     if (dev->transport == PN532_TRANSPORT_I2C) {
00269         /* PN532 I2C returns N+1 bytes: byte[0] = RDY (0x01), byte[1..N] = payload. */
00270         uint8_t tmp[PN532_I2C_RX_MAX + 1U];
00271         uint16_t to_read = (uint16_t)n + 1U;
00272         if (to_read > sizeof(tmp)) {
00273             to_read = (uint16_t)sizeof(tmp);
00274         }
00275         (void)i2c_master_receive(dev->i2c_dev, tmp, (size_t)to_read, PN532_I2C_TIMEOUT_MS);
00276         size_t payload_len = (size_t)to_read - 1U;
00277         (void)memcpy(buff, &tmp[1], payload_len);
00278     } else {
00279         uint8_t i = 0U;
00280         (void)gpio_set_level(dev->pin_cs, GPIO_LEVEL_LOW);
```

```
00281         vTaskDelay(pdMS_TO_TICKS(PN532_CS_TOGGLE_DELAY_MS));
00282         spi_write_byte(dev, PN532_SPI_DATAREAD);
00283         for (i = 0U; i < n; i++) {
00284             vTaskDelay(pdMS_TO_TICKS(PN532_BYTE_DELAY_MS));
00285             buff[i] = spi_read_byte(dev);
00286         }
00287         (void)gpio_set_level(dev->pin_cs, GPIO_LEVEL_HIGH);
00288     }
00289 }
00290
00291 static bool check_ack(pn532_t *dev)
00292 {
00293     uint8_t ackbuff[PN532_ACK_LEN];
00294     bool result;
00295     (void)memset(ackbuff, 0, sizeof(ackbuff));
00296     read_data(dev, ackbuff, PN532_ACK_LEN);
00297     result = pn532_buffer_equal(ackbuff, pn532_ack, PN532_ACK_LEN);
00298     return result;
00299 }
00300
00301 static bool pn532_buffer_equal(const uint8_t *lhs, const uint8_t *rhs, uint8_t len)
00302 {
00303     bool equal = true;
00304     uint8_t i = 0U;
00305
00306     for (i = 0U; i < len; i++) {
00307         if (lhs[i] != rhs[i]) {
00308             equal = false;
00309         }
00310     }
00311
00312     return equal;
00313 }
00314
00315 static void write_command(pn532_t *dev, const uint8_t *cmd, uint8_t cmd_len)
00316 {
00317     uint8_t frame_len = (uint8_t)(cmd_len + PN532_FRAME_TFI_OVERHEAD);
00318     uint8_t checksum  = (uint8_t)(PN532_PREAMBLE + PN532_PREAMBLE + PN532_STARTCODE2);
00319     uint8_t i = 0U;
00320
00321     if (dev->transport == PN532_TRANSPORT_I2C) {
00322         /* Build the full frame in a buffer and send it in one I2C transaction. */
00323         uint8_t frame[PN532_I2C_TX_MAX];
00324         uint16_t pos = 0U;
00325
00326         frame[pos] = PN532_PREAMBLE;   pos++;
00327         frame[pos] = PN532_PREAMBLE;   pos++;
00328         frame[pos] = PN532_STARTCODE2; pos++;
00329         frame[pos] = frame_len;        pos++;
00330         frame[pos] = (uint8_t)((uint8_t)(~frame_len) + 1U); pos++;
00331         frame[pos] = PN532_HOSTTOPN532; pos++;
00332         checksum = (uint8_t)(checksum + PN532_HOSTTOPN532);
00333
00334         for (i = 0U; i < cmd_len; i++) {
00335             frame[pos] = cmd[i]; pos++;
00336             checksum = (uint8_t)(checksum + cmd[i]);
00337         }
00338         frame[pos] = (uint8_t)(~checksum); pos++;
00339         frame[pos] = PN532_POSTAMBLE;      pos++;
00340
00341         (void)i2c_master_transmit(dev->i2c_dev, frame, (size_t)pos, PN532_I2C_TIMEOUT_MS);
00342     } else {
00343         (void)gpio_set_level(dev->pin_cs, GPIO_LEVEL_LOW);
00344         vTaskDelay(pdMS_TO_TICKS(PN532_CS_TOGGLE_DELAY_MS));
00345         spi_write_byte(dev, PN532_SPI_DATAWRITE);
00346
00347         spi_write_byte(dev, PN532_PREAMBLE);
00348         spi_write_byte(dev, PN532_PREAMBLE);
00349         spi_write_byte(dev, PN532_STARTCODE2);
00350
00351         spi_write_byte(dev, frame_len);
00352         spi_write_byte(dev, (uint8_t)((uint8_t)(~frame_len) + 1U));
00353
00354         spi_write_byte(dev, PN532_HOSTTOPN532);
00355         checksum = (uint8_t)(checksum + PN532_HOSTTOPN532);
00356
00357         for (i = 0U; i < cmd_len; i++) {
00358             spi_write_byte(dev, cmd[i]);
00359             checksum = (uint8_t)(checksum + cmd[i]);
00360         }
00361         spi_write_byte(dev, (uint8_t)(~checksum));
00362         spi_write_byte(dev, PN532_POSTAMBLE);
00363         (void)gpio_set_level(dev->pin_cs, GPIO_LEVEL_HIGH);
00364     }
00365 }
00366
00367 static bool send_command_check_ack(pn532_t *dev, const uint8_t *cmd,
```

```
00368                                   uint8_t cmd_len, uint16_t timeout)
00369 {
00370     uint16_t timer = 0U;
00371     bool timed_out = false;
00372     bool result = false;
00373
00374     write_command(dev, cmd, cmd_len);
00375
00376     /* Wait until the PN532 signals it is ready to send the ACK frame. */
00377     while ((!timed_out) && (read_ready(dev) != PN532_SPI_READY)) {
00378         if (timeout != 0U) {
00379             timer = (uint16_t)(timer + PN532_POLL_INTERVAL_MS);
00380             if (timer > timeout) {
00381                 timed_out = true;
00382             }
00383         }
00384         if (!timed_out) {
00385             vTaskDelay(pdMS_TO_TICKS(PN532_POLL_INTERVAL_MS));
00386         }
00387     }
00388
00389     if (timed_out) {
00390         ESP_LOGE(PN532_LOG_TAG, "cmd=0x%02X: TIMEOUT waiting for ACK", (unsigned)cmd[0]);
00391     } else {
00392         bool ack_ok = check_ack(dev);
00393
00394         if (ack_ok) {
00395             timer = 0U;
00396             timed_out = false;
00397
00398             /* Wait until the PN532 signals it is ready to send the response. */
00399             while ((!timed_out) && (read_ready(dev) != PN532_SPI_READY)) {
00400                 if (timeout != 0U) {
00401                     timer = (uint16_t)(timer + PN532_POLL_INTERVAL_MS);
00402                     if (timer > timeout) {
00403                         timed_out = true;
00404                     }
00405                 }
00406                 if (!timed_out) {
00407                     vTaskDelay(pdMS_TO_TICKS(PN532_POLL_INTERVAL_MS));
00408                 }
00409             }
00410
00411             if (timed_out) {
00412                 ESP_LOGE(PN532_LOG_TAG, "cmd=0x%02X: TIMEOUT waiting for response", (unsigned)cmd[0]);
00413             }
00414             result = !timed_out;
00415         } else {
00416             ESP_LOGE(PN532_LOG_TAG, "cmd=0x%02X: ACK mismatch", (unsigned)cmd[0]);
00417         }
00418     }
00419
00420     return result;
00421 }
00422
00423 /* Read one complete PN532 response frame within a single CS-low window.
00424  * Handles both normal frames (5-byte header) and extended frames (8-byte header,
00425  * indicated by frame[3]==0xFF and frame[4]==0xFF).
00426  * CS is released as soon as the last real frame byte is read, preventing
00427  * spurious clocks from corrupting the PN532 SPI state machine. */
00428 static uint16_t read_data_apdu_frame(pn532_t *dev, uint8_t *buff, uint16_t max_len)
00429 {
00430     uint16_t body_len = 0U;
00431     uint16_t total_len = 0U;
00432     bool is_extended = false;
00433
00434     if (dev->transport == PN532_TRANSPORT_I2C) {
00435         /* PN532 I²C returns 1 RDY byte + N payload bytes in a single read.
00436          * Pull the whole maximum response at once, then parse the frame
00437          * header to figure out the actual length. */
00438         uint8_t tmp[PN532_EXCHANGE_FRAME_MAX + 1U];
00439         uint16_t to_read = max_len + 1U;
00440         if (to_read > (uint16_t)sizeof(tmp)) {
00441             to_read = (uint16_t)sizeof(tmp);
00442         }
00443         (void)i2c_master_receive(dev->i2c_dev, tmp, (size_t)to_read,
00444                                  PN532_I2C_TIMEOUT_MS);
00445         size_t payload_len = (size_t)to_read - (size_t)1U;
00446         (void)memcpy(buff, &tmp[1], payload_len);
00447
00448         is_extended = ((buff[PN532_EXCHANGE_LEN_OFFSET] == PN532_EXT_FRAME_INDICATOR) &&
00449                        (buff[PN532_EXCHANGE_LEN_OFFSET + 1U] == PN532_EXT_FRAME_INDICATOR));
00450         if (is_extended) {
00451             body_len = ((uint16_t)buff[PN532_EXT_FRAME_LENHI_OFFSET] « 8U)
00452                      | (uint16_t)buff[PN532_EXT_FRAME_LENLO_OFFSET];
00453             total_len = (uint16_t)PN532_EXT_FRAME_HDR_LEN
00454                       + body_len
```

```
00455                       + (uint16_t)PN532_FRAME_TAIL_LEN;
00456         } else {
00457             body_len = (uint16_t)buff[PN532_EXCHANGE_LEN_OFFSET];
00458             total_len = (uint16_t)PN532_FRAME_HDR_LEN
00459                       + body_len
00460                       + (uint16_t)PN532_FRAME_TAIL_LEN;
00461         }
00462         if (total_len > max_len) {
00463             total_len = max_len;
00464         }
00465     } else {
00466         /* SPI path: byte-by-byte over a single CS-low window. */
00467         uint16_t i;
00468         (void)gpio_set_level(dev->pin_cs, GPIO_LEVEL_LOW);
00469         vTaskDelay(pdMS_TO_TICKS(PN532_CS_TOGGLE_DELAY_MS));
00470         spi_write_byte(dev, PN532_SPI_DATAREAD);
00471
00472         for (i = 0U; i < PN532_FRAME_HDR_LEN; i++) {
00473             vTaskDelay(pdMS_TO_TICKS(PN532_BYTE_DELAY_MS));
00474             buff[i] = spi_read_byte(dev);
00475         }
00476
00477         is_extended = ((buff[PN532_EXCHANGE_LEN_OFFSET] == PN532_EXT_FRAME_INDICATOR) &&
00478                        (buff[PN532_EXCHANGE_LEN_OFFSET + 1U] == PN532_EXT_FRAME_INDICATOR));
00479
00480         if (is_extended) {
00481             /* Read the 3 additional extended-frame header bytes: LEN_H, LEN_L, LCS. */
00482             for (i = PN532_FRAME_HDR_LEN; i < PN532_EXT_FRAME_HDR_LEN; i++) {
00483                 vTaskDelay(pdMS_TO_TICKS(PN532_BYTE_DELAY_MS));
00484                 buff[i] = spi_read_byte(dev);
00485             }
00486             body_len = ((uint16_t)buff[PN532_EXT_FRAME_LENHI_OFFSET] << 8U)
00487                      | (uint16_t)buff[PN532_EXT_FRAME_LENLO_OFFSET];
00488             total_len = (uint16_t)PN532_EXT_FRAME_HDR_LEN
00489                       + body_len
00490                       + (uint16_t)PN532_FRAME_TAIL_LEN;
00491         } else {
00492             body_len = (uint16_t)buff[PN532_EXCHANGE_LEN_OFFSET];
00493             total_len = (uint16_t)PN532_FRAME_HDR_LEN
00494                       + body_len
00495                       + (uint16_t)PN532_FRAME_TAIL_LEN;
00496         }
00497
00498         if (total_len > max_len) {
00499             total_len = max_len;
00500         }
00501
00502         i = (uint16_t)(is_extended ? PN532_EXT_FRAME_HDR_LEN : PN532_FRAME_HDR_LEN);
00503         while (i < total_len) {
00504             vTaskDelay(pdMS_TO_TICKS(PN532_BYTE_DELAY_MS));
00505             buff[i] = spi_read_byte(dev);
00506             i++;
00507         }
00508
00509         (void)gpio_set_level(dev->pin_cs, GPIO_LEVEL_HIGH);
00510     }
00511
00512     return total_len;
00513 }
00514
00515 /*****************************************************************
00516  * Transport-specific init helpers
00517  *****************************************************************/
00518
00519 static esp_err_t pn532_init_spi(pn532_t *dev, const pn532_config_t *config)
00520 {
00521     gpio_config_t io_conf;
00522     spi_bus_config_t buscfg;
00523     spi_device_interface_config_t devcfg;
00524     esp_err_t ret;
00525
00526     (void)memset(&io_conf, 0, sizeof(io_conf));
00527     io_conf.pin_bit_mask = (GPIO_PIN_BITMASK_BASE << (uint32_t)config->pin_cs);
00528     io_conf.mode = GPIO_MODE_OUTPUT;
00529     io_conf.pull_up_en = GPIO_PULLUP_DISABLE;
00530     io_conf.pull_down_en = GPIO_PULLDOWN_DISABLE;
00531     io_conf.intr_type = GPIO_INTR_DISABLE;
00532     (void)gpio_config(&io_conf);
00533     (void)gpio_set_level(config->pin_cs, GPIO_LEVEL_HIGH);
00534     dev->pin_cs = config->pin_cs;
00535
00536     if (config->skip_bus_init) {
00537         ret = ESP_OK;
00538     } else {
00539         (void)memset(&buscfg, 0, sizeof(buscfg));
00540         buscfg.mosi_io_num = config->pin_mosi;
00541         buscfg.miso_io_num = config->pin_miso;
```

```
00542         buscfg.sclk_io_num = config->pin_sclk;
00543         buscfg.quadwp_io_num = PN532_SPI_NO_PIN;
00544         buscfg.quadhd_io_num = PN532_SPI_NO_PIN;
00545         buscfg.max_transfer_sz = 0;
00546         ret = spi_bus_initialize(config->spi_host, &buscfg, SPI_DMA_CH_AUTO);
00547     }
00548
00549     if (ret == ESP_OK) {
00550         (void)memset(&devcfg, 0, sizeof(devcfg));
00551         devcfg.mode = PN532_SPI_MODE;
00552         devcfg.clock_speed_hz = (int)PN532_SPI_CLOCK_HZ;
00553         devcfg.spics_io_num = PN532_SPI_NO_PIN;
00554         devcfg.queue_size = PN532_SPI_QUEUE_SIZE;
00555         devcfg.flags = SPI_DEVICE_BIT_LSBFIRST;
00556         ret = spi_bus_add_device(config->spi_host, &devcfg, &dev->spi);
00557
00558         if ((ret != ESP_OK) && (!config->skip_bus_init)) {
00559             (void)spi_bus_free(config->spi_host);
00560         }
00561     }
00562
00563     if (ret == ESP_OK) {
00564         /* Wake-up sequence: send 0x55 0x55 + 3x preamble while CS is low. */
00565         (void)gpio_set_level(dev->pin_cs, GPIO_LEVEL_LOW);
00566         vTaskDelay(pdMS_TO_TICKS(PN532_CS_TOGGLE_DELAY_MS));
00567         spi_write_byte(dev, PN532_WAKEUP_BYTE);
00568         spi_write_byte(dev, PN532_WAKEUP_BYTE);
00569         spi_write_byte(dev, PN532_PREAMBLE);
00570         spi_write_byte(dev, PN532_PREAMBLE);
00571         spi_write_byte(dev, PN532_PREAMBLE);
00572         (void)gpio_set_level(dev->pin_cs, GPIO_LEVEL_HIGH);
00573         vTaskDelay(pdMS_TO_TICKS(PN532_WAKEUP_DELAY_MS));
00574     }
00575
00576     return ret;
00577 }
00578
00579 static esp_err_t pn532_init_i2c(pn532_t *dev, const pn532_config_t *config)
00580 {
00581     esp_err_t ret;
00582
00583     /* Optional reset pulse. */
00584     if (config->pin_rst >= 0) {
00585         gpio_config_t rst_cfg;
00586         (void)memset(&rst_cfg, 0, sizeof(rst_cfg));
00587         rst_cfg.pin_bit_mask = (GPIO_PIN_BITMASK_BASE « (uint32_t)config->pin_rst);
00588         rst_cfg.mode = GPIO_MODE_OUTPUT;
00589         (void)gpio_config(&rst_cfg);
00590         (void)gpio_set_level(config->pin_rst, GPIO_LEVEL_LOW);
00591         vTaskDelay(pdMS_TO_TICKS(50));
00592         (void)gpio_set_level(config->pin_rst, GPIO_LEVEL_HIGH);
00593         vTaskDelay(pdMS_TO_TICKS(100));
00594         dev->pin_rst = config->pin_rst;
00595     } else {
00596         dev->pin_rst = -1;
00597     }
00598
00599     /* Optional IRQ input (configured but polling is used). */
00600     if (config->pin_irq >= 0) {
00601         gpio_config_t irq_cfg;
00602         (void)memset(&irq_cfg, 0, sizeof(irq_cfg));
00603         irq_cfg.pin_bit_mask = (GPIO_PIN_BITMASK_BASE « (uint32_t)config->pin_irq);
00604         irq_cfg.mode = GPIO_MODE_INPUT;
00605         irq_cfg.pull_up_en = GPIO_PULLUP_ENABLE;
00606         (void)gpio_config(&irq_cfg);
00607         dev->pin_irq = config->pin_irq;
00608     } else {
00609         dev->pin_irq = -1;
00610     }
00611
00612     /* Create the I2C master bus (new IDF v5.x API). */
00613     i2c_master_bus_config_t bus_cfg;
00614     (void)memset(&bus_cfg, 0, sizeof(bus_cfg));
00615     bus_cfg.i2c_port = config->i2c_port;
00616     bus_cfg.sda_io_num = config->pin_sda;
00617     bus_cfg.scl_io_num = config->pin_scl;
00618     bus_cfg.clk_source = I2C_CLK_SRC_DEFAULT;
00619     bus_cfg.glitch_ignore_cnt = 7;
00620     bus_cfg.flags.enable_internal_pullup = true;
00621
00622     ret = i2c_new_master_bus(&bus_cfg, &dev->i2c_bus);
00623     if (ret != ESP_OK) {
00624         ESP_LOGE(PN532_LOG_TAG, "i2c_new_master_bus failed: %d", ret);
00625     } else {
00626         /* Attach the PN532 device on the bus. */
00627         i2c_device_config_t dev_cfg;
00628         (void)memset(&dev_cfg, 0, sizeof(dev_cfg));
```

```
00629         dev_cfg.dev_addr_length = I2C_ADDR_BIT_LEN_7;
00630         dev_cfg.device_address  = PN532_I2C_ADDRESS;
00631         dev_cfg.scl_speed_hz    = (config->i2c_clock_hz != 0U) ? config->i2c_clock_hz : 100000U;
00632
00633         ret = i2c_master_bus_add_device(dev->i2c_bus, &dev_cfg, &dev->i2c_dev);
00634         if (ret != ESP_OK) {
00635             ESP_LOGE(PN532_LOG_TAG, "i2c_master_bus_add_device failed: %d", ret);
00636             (void)i2c_del_master_bus(dev->i2c_bus);
00637             dev->i2c_bus = NULL;
00638         } else {
00639             /* Adafruit_PN532::begin does a small delay between bus init
00640              * and the first I2C transaction; mirror it. */
00641             vTaskDelay(pdMS_TO_TICKS(10));
00642
00643             ESP_LOGI(PN532_LOG_TAG,
00644                      "I2C master bus installed on port %d (SDA=%d SCL=%d %lu Hz)",
00645                      config->i2c_port, (int)config->pin_sda, (int)config->pin_scl,
00646                      (unsigned long)dev_cfg.scl_speed_hz);
00647         }
00648     }
00649
00650     return ret;
00651 }
00652
00653 /*****************************************************************
00654  * Public API
00655  *****************************************************************/
00656
00657 esp_err_t pn532_init(pn532_t *dev, const pn532_config_t *config)
00658 {
00659     esp_err_t ret;
00660
00661     (void)memset(dev, 0, sizeof(*dev));
00662     dev->transport = config->transport;
00663
00664     if (config->transport == PN532_TRANSPORT_I2C) {
00665         ret = pn532_init_i2c(dev, config);
00666     } else {
00667         ret = pn532_init_spi(dev, config);
00668     }
00669
00670     if (ret == ESP_OK) {
00671         /* Transport-specific post-wakeup sequencing.
00672          *
00673          * I2C -- Soft-Power-Down recovery:
00674          *   When the host soft-reboots with the PN532 still powered the chip
00675          *   enters Soft-Power-Down.  The main I2C peripheral is OFF; a
00676          *   dedicated wake-up watcher listens on the bus.  The first
00677          *   transaction that matches the slave address triggers the watcher,
00678          *   but the firmware must then disable the watcher and re-enable the
00679          *   main I2C peripheral -- a process that takes up to 500 ms and
00680          *   DISCARDS the triggering frame.  We therefore send a sacrificial
00681          *   SAMConfig first, wait 500 ms, then send the real SAMConfig.
00682          *
00683          * SPI -- post-power-on synchronisation:
00684          *   The Adafruit_PN532 library always issues SAMConfig as the very
00685          *   first functional command after wakeup.  On some PN532 modules the
00686          *   chip's internal power-on sequencing is not complete until roughly
00687          *   2 s after VCC is applied; a SAMConfig sent before that point is
00688          *   silently ignored (no ACK -- 1 s command timeout).  By issuing a
00689          *   sacrificial SAMConfig here we absorb that 1 s stall inside
00690          *   pn532_init so that the application-level SAMConfig issued by
00691          *   wallet.begin() arrives after the chip is fully ready. */
00692         if (config->transport == PN532_TRANSPORT_I2C) {
00693             ESP_LOGI(PN532_LOG_TAG, "I2C wake-up trigger (sacrificial SAMConfig)");
00694             (void)pn532_sam_config(dev);
00695             vTaskDelay(pdMS_TO_TICKS(500));
00696             ESP_LOGI(PN532_LOG_TAG, "Real SAMConfig after wake-up");
00697             (void)pn532_sam_config(dev);
00698             vTaskDelay(pdMS_TO_TICKS(PN532_SYNC_DELAY_MS));
00699         } else {
00700             ESP_LOGI(PN532_LOG_TAG, "SPI post-power-on SAMConfig (sacrificial)");
00701             (void)pn532_sam_config(dev);
00702             vTaskDelay(pdMS_TO_TICKS(PN532_SYNC_DELAY_MS));
00703         }
00704
00705         /* Read the firmware version to confirm the chip is alive and to drain
00706          * any stale response bytes from the PN532's output FIFO before the
00707          * application-level SAMConfig is issued. */
00708         (void)pn532_get_firmware_version(dev);
00709         vTaskDelay(pdMS_TO_TICKS(PN532_SYNC_DELAY_MS));
00710
00711         ESP_LOGI(PN532_LOG_TAG, "PN532 initialized (%s)",
00712                  (config->transport == PN532_TRANSPORT_I2C) ? "I2C" : "SPI");
00713     }
00714
00715     return ret;
```

```
00716 }
00717
00718 uint32_t pn532_get_firmware_version(pn532_t *dev)
00719 {
00720     uint8_t pn532_packetbuffer[PN532_FIRMWARE_RESP_LEN];
00721     uint32_t response = 0U;
00722     bool ack_received = false;
00723
00724     (void)memset(pn532_packetbuffer, 0, sizeof(pn532_packetbuffer));
00725     pn532_packetbuffer[0] = PN532_FIRMWAREVERSION;
00726     ack_received = send_command_check_ack(dev, pn532_packetbuffer,
00727                                           PN532_FIRMWARE_CMD_LEN, PN532_CMD_TIMEOUT_MS);
00728
00729     if (!ack_received) {
00730         ESP_LOGE(PN532_LOG_TAG, "No ACK from PN532");
00731     } else {
00732         read_data(dev, pn532_packetbuffer, PN532_FIRMWARE_RESP_LEN);
00733         ESP_LOG_BUFFER_HEX_LEVEL(PN532_LOG_TAG, pn532_packetbuffer,
00734                                  PN532_FIRMWARE_RESP_LEN, ESP_LOG_INFO);
00735
00736         if (!pn532_buffer_equal(pn532_packetbuffer, pn532_response_fw, PN532_FIRMWARE_HDR_LEN)) {
00737             ESP_LOGE(PN532_LOG_TAG, "Unexpected firmware response");
00738         } else {
00739             response = (uint32_t)pn532_packetbuffer[PN532_FW_IC_OFFSET];
00740             response <<= PN532_BYTE_SHIFT_BITS;
00741             response |= (uint32_t)pn532_packetbuffer[PN532_FW_VER_OFFSET];
00742             response <<= PN532_BYTE_SHIFT_BITS;
00743             response |= (uint32_t)pn532_packetbuffer[PN532_FW_REV_OFFSET];
00744             response <<= PN532_BYTE_SHIFT_BITS;
00745             response |= (uint32_t)pn532_packetbuffer[PN532_FW_SUPPORT_OFFSET];
00746         }
00747     }
00748
00749     return response;
00750 }
00751
00752 bool pn532_sam_config(pn532_t *dev)
00753 {
00754     uint8_t pn532_packetbuffer[PN532_SAM_RESP_LEN];
00755     bool ack_received = false;
00756     bool result = false;
00757
00758     (void)memset(pn532_packetbuffer, 0, sizeof(pn532_packetbuffer));
00759     pn532_packetbuffer[0] = PN532_SAMCONFIGURATION;
00760     pn532_packetbuffer[1] = PN532_SAM_NORMAL_MODE;
00761     pn532_packetbuffer[2] = PN532_SAM_TIMEOUT;
00762     pn532_packetbuffer[3] = PN532_SAM_USE_IRQ;
00763
00764     ack_received = send_command_check_ack(dev, pn532_packetbuffer,
00765                                           PN532_SAM_CMD_LEN, PN532_CMD_TIMEOUT_MS);
00766
00767     if (ack_received) {
00768         read_data(dev, pn532_packetbuffer, PN532_SAM_RESP_LEN);
00769         result = (pn532_packetbuffer[PN532_SAM_RESP_CODE_OFFSET] == PN532_SAM_RESP_CODE);
00770     }
00771
00772     return result;
00773 }
00774
00775 uint32_t pn532_read_passive_target_id(pn532_t *dev, uint8_t cardbaudrate)
00776 {
00777     uint8_t pn532_packetbuffer[PN532_PASSIVE_RESP_LEN];
00778     uint32_t cid = 0U;
00779     bool ack_received = false;
00780
00781     (void)memset(pn532_packetbuffer, 0, sizeof(pn532_packetbuffer));
00782     pn532_packetbuffer[0] = PN532_INLISTPASSIVETARGET;
00783     pn532_packetbuffer[1] = PN532_PASSIVE_MAX_TARGETS;
00784     pn532_packetbuffer[2] = cardbaudrate;
00785
00786     ack_received = send_command_check_ack(dev, pn532_packetbuffer,
00787                                           PN532_PASSIVE_CMD_LEN, PN532_CMD_TIMEOUT_MS);
00788
00789     if (ack_received) {
00790         read_data(dev, pn532_packetbuffer, PN532_PASSIVE_RESP_LEN);
00791
00792         if (pn532_packetbuffer[PN532_PASSIVE_NUM_TARGETS_OFFSET] == PN532_PASSIVE_EXPECTED_TARGETS) {
00793             uint8_t uid_len = pn532_packetbuffer[PN532_PASSIVE_UID_LEN_OFFSET];
00794             uint8_t i = 0U;
00795             for (i = 0U; i < uid_len; i++) {
00796                 cid <<= PN532_BYTE_SHIFT_BITS;
00797                 cid |= (uint32_t)pn532_packetbuffer[PN532_PASSIVE_UID_DATA_OFFSET + i];
00798             }
00799         }
00800     }
00801
00802     return cid;
```

```
00803 }
00804
00805 bool pn532_send_apdu(pn532_t *dev, const uint8_t *apdu, uint8_t apdu_len,
00806                      uint8_t *response, uint16_t *response_len)
00807 {
00808     uint8_t cmd[PN532_MAX_APDU_LEN + PN532_EXCHANGE_CMD_OVERHEAD];
00809     uint8_t frame[PN532_EXCHANGE_FRAME_MAX];
00810     bool result = false;
00811
00812     (void)memset(cmd, 0, sizeof(cmd));
00813     (void)memset(frame, 0, sizeof(frame));
00814
00815     if (apdu_len <= PN532_MAX_APDU_LEN) {
00816         uint8_t cmd_total_len = 0U;
00817         bool ack_received = false;
00818
00819         cmd[0] = PN532_INDATAEXCHANGE;
00820         cmd[1] = PN532_EXCHANGE_TG;
00821         (void)memcpy(&cmd[PN532_EXCHANGE_CMD_OVERHEAD], apdu, apdu_len);
00822         cmd_total_len = (uint8_t)(apdu_len + PN532_EXCHANGE_CMD_OVERHEAD);
00823
00824         ack_received = send_command_check_ack(dev, cmd, cmd_total_len, PN532_APDU_TIMEOUT_MS);
00825
00826         if (ack_received) {
00827             (void)read_data_apdu_frame(dev, frame, (uint16_t)PN532_EXCHANGE_FRAME_MAX);
00828
00829             bool is_extended = ((frame[PN532_EXCHANGE_LEN_OFFSET] == PN532_EXT_FRAME_INDICATOR) &&
00830                                 (frame[PN532_EXCHANGE_LEN_OFFSET + 1U] == PN532_EXT_FRAME_INDICATOR));
00831
00832             uint8_t err_byte = 0U;
00833             uint16_t data_offset = 0U;
00834             uint16_t data_len = 0U;
00835
00836             if (is_extended) {
00837                 uint16_t body_len = ((uint16_t)frame[PN532_EXT_FRAME_LENHI_OFFSET] « 8U)
00838                                     | (uint16_t)frame[PN532_EXT_FRAME_LENLO_OFFSET];
00839                 err_byte    = frame[PN532_EXT_EXCHANGE_ERR_OFFSET];
00840                 data_offset = (uint16_t)PN532_EXT_EXCHANGE_DATA_OFFSET;
00841                 data_len    = (body_len >= (uint16_t)PN532_EXCHANGE_LEN_BIAS)
00842                               ? (body_len - (uint16_t)PN532_EXCHANGE_LEN_BIAS)
00843                               : 0U;
00844             } else {
00845                 uint8_t len_field = frame[PN532_EXCHANGE_LEN_OFFSET];
00846                 err_byte    = frame[PN532_EXCHANGE_STATUS_OFFSET];
00847                 data_offset = (uint16_t)PN532_EXCHANGE_DATA_OFFSET;
00848                 if (len_field >= (uint8_t)PN532_EXCHANGE_LEN_BIAS) {
00849                     uint8_t raw_data_len = (uint8_t)(len_field - (uint8_t)PN532_EXCHANGE_LEN_BIAS);
00850                     data_len = (uint16_t)raw_data_len;
00851                 } else {
00852                     data_len = 0U;
00853                 }
00854             }
00855
00856             if (err_byte == PN532_EXCHANGE_STATUS_OK) {
00857                 uint16_t buf_cap = *response_len;
00858                 uint16_t copy_len = (data_len <= buf_cap) ? data_len : buf_cap;
00859
00860                 (void)memcpy(response, &frame[data_offset], (size_t)copy_len);
00861                 *response_len = copy_len;
00862                 result = true;
00863             } else {
00864                 ESP_LOGE(PN532_LOG_TAG, "APDU INS=0x%02X: ERR=0x%02X",
00865                          (unsigned)apdu[1], (unsigned)err_byte);
00866             }
00867         } else {
00868             ESP_LOGE(PN532_LOG_TAG, "APDU INS=0x%02X: no ACK", (unsigned)apdu[1]);
00869         }
00870     }
00871
00872     return result;
00873 }
00874
00875 bool pn532_release_target(pn532_t *dev)
00876 {
00877     uint8_t cmd[PN532_INRELEASE_CMD_LEN];
00878     uint8_t resp[PN532_INRELEASE_RESP_LEN];
00879     bool ack_received = false;
00880     bool result = false;
00881
00882     (void)memset(cmd, 0, sizeof(cmd));
00883     (void)memset(resp, 0, sizeof(resp));
00884
00885     cmd[0] = PN532_INRELEASE;
00886     cmd[1] = PN532_EXCHANGE_TG;
00887
00888     ack_received = send_command_check_ack(dev, cmd, PN532_INRELEASE_CMD_LEN, PN532_CMD_TIMEOUT_MS);
00889
```

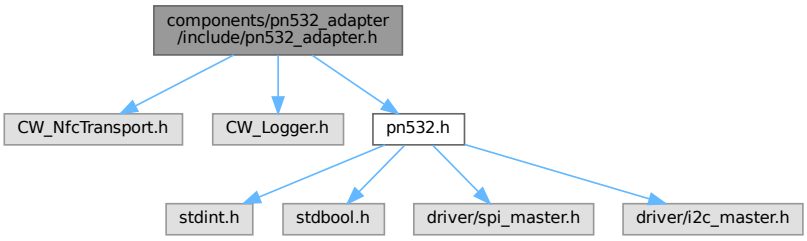
```
00890     if (ack_received) {
00891         read_data(dev, resp, PN532_INRELEASE_RESP_LEN);
00892         result = (resp[PN532_EXCHANGE_STATUS_OFFSET] == PN532_EXCHANGE_STATUS_OK);
00893     }
00894
00895     return result;
00896 }
```

5.21 components/pn532_adapter/include/pn532_adapter.h File Reference

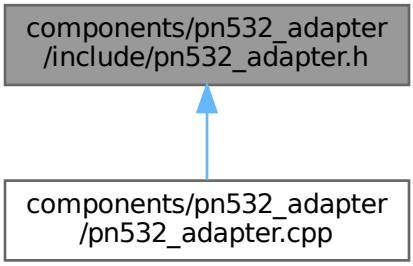
Self-contained CW_NfcTransport adapter that owns its pn532_t handle.

```
#include "CW_NfcTransport.h"
#include "CW_Logger.h"
#include "pn532.h"
```

Include dependency graph for pn532_adapter.h:



This graph shows which files directly or indirectly include this file:



Classes

- class PN532Adapter

  Self-contained CW_NfcTransport that owns its pn532_t instance.

5.21.1 Detailed Description

Self-contained CW_NfcTransport adapter that owns its pn532_t handle.

PN532Adapter is an alternative to Pn532NfcTransport for callers that prefer to hand a pn532_config_t to the adapter and let it manage the full driver lifetime internally, rather than constructing the pn532_t externally.
Definition in file pn532_adapter.h.

5.22 pn532_adapter.h

Go to the documentation of this file.

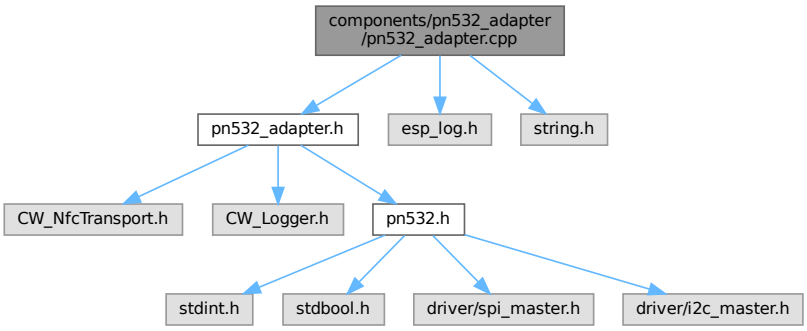
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00017
00018 #ifndef PN532_ADAPTER_H
00019 #define PN532_ADAPTER_H
00020
00021 #include "CW_NfcTransport.h"
00022 #include "CW_Logger.h"
00023 #include "pn532.h"
00024
00054 class PN532Adapter : public CW_NfcTransport {
00055 public:
00066     PN532Adapter(const pn532_config_t &config, CW_Logger &logger);
00067
00078     bool begin() override;
00079
00086     bool inListPassiveTarget() override;
00087
00101     bool sendAPDU(const uint8_t *apdu, uint8_t apduLen,
00102                   uint8_t *response, uint8_t &responseLen) override;
00103
00119     bool sendAPDULarge(const uint8_t *apdu, uint8_t apduLen,
00120                        uint8_t *response, uint16_t &responseLen) override;
00121
00128     void resetReader() override;
00129
00138     bool printFirmwareVersion() override;
00139
00140 private:
00141     pn532_config_t  _config;
00142     pn532_t         _dev;
00143     CW_Logger      &_logger;
00144     bool            _initialized;
00145 };
00146
00147 #endif /* PN532_ADAPTER_H */
```

5.23 components/pn532_adapter/pn532_adapter.cpp File Reference

Implementation of PN532Adapter — self-contained PN532 NFC transport.

```
#include "pn532_adapter.h"
#include "esp_log.h"
#include <string.h>
```

Include dependency graph for pn532_adapter.cpp:



Variables

- static const char * TAG = "pn532_adapter"
- static const uint32_t FW_IC_SHIFT = 24U
- static const uint32_t FW_VER_SHIFT = 16U
- static const uint32_t FW_REV_SHIFT = 8U
- static const uint32_t FW_BYTE_MASK = 0xFFU

5.23.1 Detailed Description

Implementation of PN532Adapter — self-contained PN532 NFC transport.
Owns a pn532_t handle internally and initialises it on the first PN532Adapter::begin call. Full API documentation lives on the declarations in pn532_adapter.h.
Definition in file pn532_adapter.cpp.

5.23.2 Variable Documentation

5.23.2.1 FW_BYTE_MASK

```
const uint32_t FW_BYTE_MASK = 0xFFU  [static]
```

Definition at line 25 of file pn532_adapter.cpp.
Referenced by PN532Adapter::printFirmwareVersion().

5.23.2.2 FW_IC_SHIFT

```
const uint32_t FW_IC_SHIFT = 24U  [static]
```

Definition at line 22 of file pn532_adapter.cpp.
Referenced by PN532Adapter::printFirmwareVersion().

5.23.2.3 FW_REV_SHIFT

```
const uint32_t FW_REV_SHIFT = 8U  [static]
```

Definition at line 24 of file pn532_adapter.cpp.
Referenced by PN532Adapter::printFirmwareVersion().

5.23.2.4 FW_VER_SHIFT

```
const uint32_t FW_VER_SHIFT = 16U  [static]
```

Definition at line 23 of file pn532_adapter.cpp.
Referenced by PN532Adapter::printFirmwareVersion().

5.23.2.5 TAG

```
const char* TAG = "pn532_adapter"  [static]
```

Definition at line 19 of file pn532_adapter.cpp.

5.24 pn532_adapter.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00014
00015 #include "pn532_adapter.h"
00016 #include "esp_log.h"
00017 #include <string.h>
00018
00019 static const char *TAG = "pn532_adapter";
00020
00021 /* Bit positions used to extract firmware version fields from the packed uint32_t. */
00022 static const uint32_t FW_IC_SHIFT = 24U;
00023 static const uint32_t FW_VER_SHIFT = 16U;
00024 static const uint32_t FW_REV_SHIFT = 8U;
00025 static const uint32_t FW_BYTE_MASK = 0xFFU;
00026
00027 PN532Adapter::PN532Adapter(const pn532_config_t &config, CW_Logger &logger)
00028     : _config(config), _dev{}, _logger(logger), _initialized(false)
00029 {
00030 }
00031
00032 bool PN532Adapter::begin()
00033 {
00034     esp_err_t ret = ESP_FAIL;
00035     bool result = false;
00036
00037     ret = pn532_init(&_dev, &_config);
00038     if (ret != ESP_OK) {
00039         ESP_LOGE(TAG, "pn532_init failed: %s", esp_err_to_name(ret));
00040     } else {
00041         _initialized = true;
00042         result = true;
00043     }
00044
00045     return result;
00046 }
00047
00048 bool PN532Adapter::inListPassiveTarget()
00049 {
00050     uint32_t uid = 0U;
00051     bool result = false;
00052
00053     if (_initialized) {
00054         uid = pn532_read_passive_target_id(&_dev, PN532_MIFARE_ISO14443A);
00055         result = (uid != 0U);
00056     }
00057
00058     return result;
00059 }
00060
00061 bool PN532Adapter::sendAPDU(const uint8_t *apdu, uint8_t apduLen,
00062                             uint8_t *response, uint8_t &responseLen)
00063 {
00064     bool result = false;
00065
00066     if (_initialized) {
00067         uint16_t len = static_cast<uint16_t>(responseLen);
00068         result = pn532_send_apdu(&_dev, apdu, apduLen, response, &len);
00069         responseLen = static_cast<uint8_t>((len > static_cast<uint16_t>(UINT8_MAX))
00070                                            ? static_cast<uint16_t>(UINT8_MAX) : len);
00071     }
00072
00073     return result;
00074 }
00075
00076 bool PN532Adapter::sendAPDULarge(const uint8_t *apdu, uint8_t apduLen,
00077                                  uint8_t *response, uint16_t &responseLen)
00078 {
00079     bool result = false;
00080
00081     if (_initialized) {
00082         result = pn532_send_apdu(&_dev, apdu, apduLen, response, &responseLen);
```

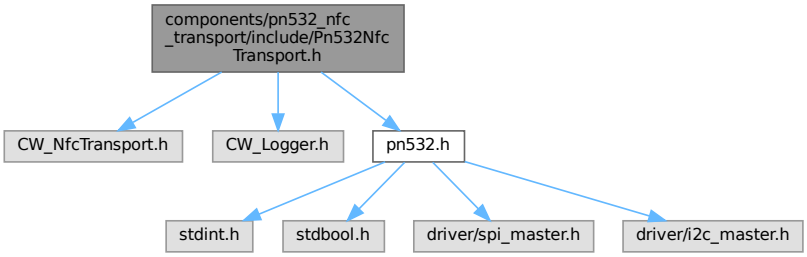
```
00083     }
00084
00085     return result;
00086 }
00087
00088 void PN532Adapter::resetReader()
00089 {
00090     if (_initialized) {
00091         (void)pn532_release_target(&_dev);
00092     }
00093 }
00094
00095 bool PN532Adapter::printFirmwareVersion()
00096 {
00097     uint32_t version = 0U;
00098     bool result = false;
00099
00100     if (_initialized) {
00101         version = pn532_get_firmware_version(&_dev);
00102         if (version == 0U) {
00103             _logger.println("PN532 firmware query failed");
00104         } else {
00105             _logger.print("PN5");
00106             _logger.print(static_cast<uint8_t>((version » FW_IC_SHIFT) & FW_BYTE_MASK), HEX);
00107             _logger.print(" Firmware v");
00108             _logger.print(static_cast<uint8_t>((version » FW_VER_SHIFT) & FW_BYTE_MASK), DEC);
00109             _logger.print(".");
00110             _logger.println(static_cast<uint8_t>((version » FW_REV_SHIFT) & FW_BYTE_MASK), DEC);
00111             result = true;
00112         }
00113     }
00114
00115     return result;
00116 }
```

5.25 components/pn532_nfc_transport/include/Pn532NfcTransport.h File Reference

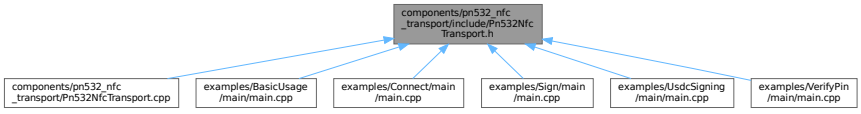
CW_NfcTransport adapter wrapping the ESP-IDF PN532 NFC driver.

```
#include "CW_NfcTransport.h"
#include "CW_Logger.h"
#include "pn532.h"
```

Include dependency graph for Pn532NfcTransport.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Pn532NfcTransport

  CW_NfcTransport implementation backed by the ESP-IDF PN532 driver.

5.25.1 Detailed Description

CW_NfcTransport adapter wrapping the ESP-IDF PN532 NFC driver.
Pn532NfcTransport bridges the C-level PN532 NFC driver with the C++ CW_NfcTransport interface consumed by CW_SecureChannel and CryptnoxWallet. Callers construct the PN532 handle independently (via pn532_init) and inject the pointer at construction time.

Typical usage

```
pn532_t nfc;
pn532_config_t cfg = {};
cfg.transport     = PN532_TRANSPORT_SPI;
cfg.spi_host      = SPI2_HOST;
cfg.pin_cs        = 10;
cfg.skip_bus_init = true;
ESP_ERROR_CHECK(pn532_init(&nfc, &cfg));

ESP32Logger          logger;
Pn532NfcTransport    transport(&nfc, logger);
ESP32CryptoProvider  crypto;
ESP32Platform        platform;
CryptnoxWallet       wallet(transport, logger, crypto, platform);
```

Definition in file Pn532NfcTransport.h.

5.26 Pn532NfcTransport.h

Go to the documentation of this file.

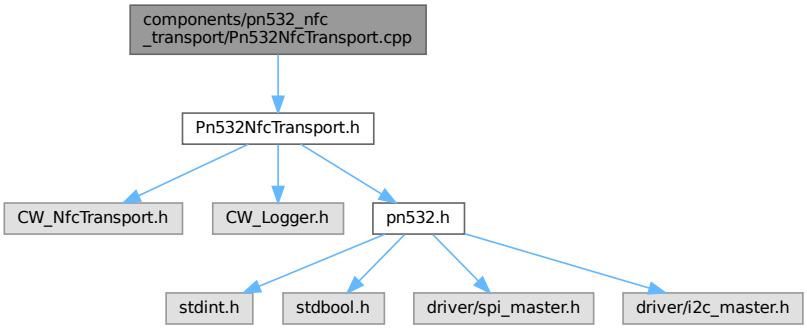
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00049
00050 #pragma once
00051
00052 #include "CW_NfcTransport.h"
00053 #include "CW_Logger.h"
00054
00055 extern "C" {
00056 #include "pn532.h"
00057 }
00058
00080 class Pn532NfcTransport : public CW_NfcTransport {
00081 public:
00089     Pn532NfcTransport(pn532_t *dev, CW_Logger &logger);
00090
00100     bool begin() override;
00101
00111     bool inListPassiveTarget() override;
00112
00130     bool sendAPDU(const uint8_t *apdu, uint8_t apduLen,
00131                   uint8_t *response, uint8_t &responseLen) override;
00132
00150     bool sendAPDULarge(const uint8_t *apdu, uint8_t apduLen,
00151                        uint8_t *response, uint16_t &responseLen) override;
00152
00159     void resetReader() override;
00160
00170     bool printFirmwareVersion() override;
00171
00173     ~Pn532NfcTransport() override {}
00174
00175 private:
00176     pn532_t   *m_dev;
00177     CW_Logger &m_logger;
00178 };
```

5.27 components/pn532_nfc_transport/Pn532NfcTransport.cpp File Reference

Implementation of Pn532NfcTransport — ESP-IDF PN532 NFC transport.

```
#include "Pn532NfcTransport.h"
```

Include dependency graph for Pn532NfcTransport.cpp:



Variables

- static const uint32_t PN532_FW_IC_SHIFT = 24U
- static const uint32_t PN532_FW_VER_SHIFT = 16U
- static const uint32_t PN532_FW_REV_SHIFT = 8U
- static const uint32_t PN532_BYTE_MASK = 0xFFU

5.27.1 Detailed Description

Implementation of Pn532NfcTransport — ESP-IDF PN532 NFC transport.
Thin delegation layer: every method forwards directly to the corresponding PN532 NFC driver function.
Full API documentation lives on the declarations in Pn532NfcTransport.h.
Definition in file Pn532NfcTransport.cpp.

5.27.2 Variable Documentation

5.27.2.1 PN532_BYTE_MASK

```
const uint32_t PN532_BYTE_MASK = 0xFFU  [static]
```

Definition at line 20 of file Pn532NfcTransport.cpp.
Referenced by Pn532NfcTransport::printFirmwareVersion().

5.27.2.2 PN532_FW_IC_SHIFT

```
const uint32_t PN532_FW_IC_SHIFT = 24U  [static]
```

Definition at line 17 of file Pn532NfcTransport.cpp.
Referenced by Pn532NfcTransport::printFirmwareVersion().

5.27.2.3 PN532_FW_REV_SHIFT

```
const uint32_t PN532_FW_REV_SHIFT = 8U  [static]
```

Definition at line 19 of file Pn532NfcTransport.cpp.
Referenced by Pn532NfcTransport::printFirmwareVersion().

5.27.2.4 PN532_FW_VER_SHIFT

```
const uint32_t PN532_FW_VER_SHIFT = 16U  [static]
```

Definition at line 18 of file Pn532NfcTransport.cpp.
Referenced by Pn532NfcTransport::printFirmwareVersion().

5.28 Pn532NfcTransport.cpp

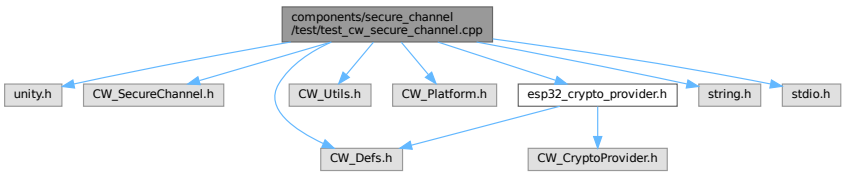
Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00014
00015 #include "Pn532NfcTransport.h"
00016
00017 static const uint32_t PN532_FW_IC_SHIFT  = 24U;
00018 static const uint32_t PN532_FW_VER_SHIFT = 16U;
00019 static const uint32_t PN532_FW_REV_SHIFT = 8U;
00020 static const uint32_t PN532_BYTE_MASK    = 0xFFU;
00021
00022 Pn532NfcTransport::Pn532NfcTransport(pn532_t *dev, CW_Logger &logger)
00023     : m_dev(dev), m_logger(logger)
00024 {
00025 }
00026
00027 bool Pn532NfcTransport::begin()
00028 {
00029     bool result = pn532_sam_config(m_dev);
00030     return result;
00031 }
00032
00033 bool Pn532NfcTransport::inListPassiveTarget()
00034 {
00035     bool result = (pn532_read_passive_target_id(m_dev, PN532_MIFARE_ISO14443A) != 0U);
00036     return result;
00037 }
00038
00039 bool Pn532NfcTransport::sendAPDU(const uint8_t *apdu, uint8_t apduLen,
00040                                  uint8_t *response, uint8_t &responseLen)
00041 {
00042     uint16_t len = static_cast<uint16_t>(responseLen);
00043     bool result = pn532_send_apdu(m_dev, apdu, apduLen, response, &len);
00044     responseLen = static_cast<uint8_t>((len > static_cast<uint16_t>(UINT8_MAX))
00045                                        ? static_cast<uint16_t>(UINT8_MAX) : len);
00046     return result;
00047 }
00048
00049 bool Pn532NfcTransport::sendAPDULarge(const uint8_t *apdu, uint8_t apduLen,
00050                                       uint8_t *response, uint16_t &responseLen)
00051 {
00052     bool result = pn532_send_apdu(m_dev, apdu, apduLen, response, &responseLen);
00053     return result;
00054 }
00055
00056 void Pn532NfcTransport::resetReader()
00057 {
00058     (void)pn532_release_target(m_dev);
00059 }
00060
00061 bool Pn532NfcTransport::printFirmwareVersion()
00062 {
00063     uint32_t version = pn532_get_firmware_version(m_dev);
00064     bool result = (version != 0U);
00065
00066     if (result) {
00067         m_logger.print("PN5");
00068         m_logger.print(static_cast<uint8_t>((version » PN532_FW_IC_SHIFT) & PN532_BYTE_MASK), HEX);
00069         m_logger.print(" firmware v");
00070         m_logger.print(static_cast<uint8_t>((version » PN532_FW_VER_SHIFT) & PN532_BYTE_MASK));
00071         m_logger.print('.');
00072         m_logger.println(static_cast<uint8_t>((version » PN532_FW_REV_SHIFT) & PN532_BYTE_MASK));
00073     }
00074
00075     return result;
00076 }
```

5.29 components/secure_channel/test/test_cw_secure_channel.cpp File Reference

```
#include "unity.h"
#include "CW_SecureChannel.h"
#include "CW_Defs.h"
#include "CW_Utils.h"
#include "CW_Platform.h"
#include "esp32_crypto_provider.h"
#include <string.h>
#include <stdio.h>
```

Include dependency graph for test_cw_secure_channel.cpp:



Classes

- class MockLogger
- class MockPlatform
- struct MockScriptEntry
- class ScriptedMockNfcTransport
- class ReflectiveMockNfcTransport

Macros

- #define SC_AES_BLOCK_BYTES (16U)
- #define SC_AES_KEY_BYTES (32U)
- #define SC_EC_COORD_BYTES (32U)
- #define SC_EC_PUBKEY_BYTES (64U)
- #define SC_CERT_MARKER_BYTES (1U)
- #define SC_CERT_NONCE_BYTES (8U)
- #define SC_CERT_KEY65_BYTES (65U)
- #define SC_CERT_TOTAL_BYTES (74U)
- #define SC_CERT_KEY_OFFSET (SC_CERT_MARKER_BYTES + SC_CERT_NONCE_BYTES)
- #define SC_SW_BYTES (2U)
- #define SC_SELECT_RESP_BYTES (26U)
- #define SC_GET_CERT_RESP_BYTES (148U)
- #define SC_OPEN_SC_RESP_BYTES (34U)
- #define SC_MUTUAL_AUTH_RESP_BYTES (66U)
- #define SC_SALT_BYTES (32U)
- #define SC_IV_BYTES (16U)
- #define SC_SHA512_OUT_BYTES (64U)
- #define SC_APDU_HEADER_LEN (4U)
- #define SC_APDU_LC_OFFSET (4U)
- #define SC_APDU_MAC_OFFSET (5U) /* header[4] + Lc[1] */
- #define SC_APDU_MAC_BYTES (16U)
- #define SC_CARD_RESP_PAYLOAD_BYTES (4U)

- #define SC_CARD_RESP_TOTAL_BYTES (SC_CARD_RESP_PAYLOAD_BYTES + SC_SW_BYTES)
- #define MOCK_MAX_SCRIPTS (8U)
- #define MOCK_MAX_RESP_BYTES (255U)
- #define MOCK_UART_BAUD_RATE (115200UL)

Functions

- TEST_CASE ("checkStatusWord: SW 0x9000 returns true", "[secure_channel]")
- TEST_CASE ("checkStatusWord: SW mismatch returns false", "[secure_channel]")
- TEST_CASE ("checkStatusWord: response shorter than 2 bytes returns false", "[secure_channel]")
- TEST_CASE ("extractCardEphemeralKey: extracts 64-byte key from synthetic certificate", "[secure_channel]")
- TEST_CASE ("extractCardEphemeralKey: null cert pointer returns false", "[secure_channel]")
- TEST_CASE ("selectApdu: succeeds when transport returns SW 0x9000", "[secure_channel]")
- TEST_CASE ("selectApdu: fails when transport returns error SW", "[secure_channel]")
- TEST_CASE ("getCardCertificate: extracts 146 certificate bytes from mock response", "[secure_channel]")
- TEST_CASE ("openSecureChannel: extracts 32-byte salt from mock response", "[secure_channel]")
- TEST_CASE ("mutuallyAuthenticate: sets session IV to first 16 bytes of mock response", "[secure_channel]")
- TEST_CASE ("key derivation: ECDH + SHA-512 split yields distinct Kenc and Kmac", "[secure_channel]")
- TEST_CASE ("aesCbcEncrypt/aesCbcDecrypt: round-trip via reflective mock returns card payload", "[secure_channel]")

Variables

- static const uint8_t K_CARD_EPHEMERAL_PUB [SC_EC_PUBKEY_BYTES]
- static const uint8_t K_TEST_KENC [SC_AES_KEY_BYTES]
- static const uint8_t K_TEST_KMAC [SC_AES_KEY_BYTES]
- static const uint8_t K_CARD_RESP_PLAINTEXT [SC_CARD_RESP_TOTAL_BYTES]
- static ESP32CryptoProvider s_crypto
- static MockLogger s_logger
- static MockPlatform s_platform
- static ScriptedMockNfcTransport s_scriptedTransport
- static ReflectiveMockNfcTransport s_reflectiveTransport

5.29.1 Macro Definition Documentation

5.29.1.1 MOCK_MAX_RESP_BYTES

```
#define MOCK_MAX_RESP_BYTES (255U)
```

Definition at line 53 of file test_cw_secure_channel.cpp.

5.29.1.2 MOCK_MAX_SCRIPTS

```
#define MOCK_MAX_SCRIPTS (8U)
```

Definition at line 52 of file test_cw_secure_channel.cpp.
Referenced by ScriptedMockNfcTransport::addScript().

5.29.1.3 MOCK_UART_BAUD_RATE

```
#define MOCK_UART_BAUD_RATE (115200UL)
```
Definition at line 54 of file test_cw_secure_channel.cpp.
Referenced by MockLogger::begin().

5.29.1.4 SC_AES_BLOCK_BYTES

```
#define SC_AES_BLOCK_BYTES (16U)
```
Definition at line 19 of file test_cw_secure_channel.cpp.
Referenced by ReflectiveMockNfcTransport::sendAPDU().

5.29.1.5 SC_AES_KEY_BYTES

```
#define SC_AES_KEY_BYTES (32U)
```
Definition at line 20 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.29.1.6 SC_APDU_HEADER_LEN

```
#define SC_APDU_HEADER_LEN (4U)
```
Definition at line 42 of file test_cw_secure_channel.cpp.

5.29.1.7 SC_APDU_LC_OFFSET

```
#define SC_APDU_LC_OFFSET (4U)
```
Definition at line 43 of file test_cw_secure_channel.cpp.

5.29.1.8 SC_APDU_MAC_BYTES

```
#define SC_APDU_MAC_BYTES (16U)
```
Definition at line 45 of file test_cw_secure_channel.cpp.
Referenced by ReflectiveMockNfcTransport::sendAPDU().

5.29.1.9 SC_APDU_MAC_OFFSET

```
#define SC_APDU_MAC_OFFSET (5U) /* header[4] + Lc[1] */
```
Definition at line 44 of file test_cw_secure_channel.cpp.
Referenced by ReflectiveMockNfcTransport::sendAPDU().

5.29.1.10 SC_CARD_RESP_PAYLOAD_BYTES

```
#define SC_CARD_RESP_PAYLOAD_BYTES (4U)
```
Definition at line 48 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.1.11 SC_CARD_RESP_TOTAL_BYTES

```
#define SC_CARD_RESP_TOTAL_BYTES (SC_CARD_RESP_PAYLOAD_BYTES + SC_SW_BYTES)
```
Definition at line 49 of file test_cw_secure_channel.cpp.

5.29.1.12 SC_CERT_KEY65_BYTES

```
#define SC_CERT_KEY65_BYTES (65U)
```
Definition at line 27 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.1.13 SC_CERT_KEY_OFFSET

```
#define SC_CERT_KEY_OFFSET (SC_CERT_MARKER_BYTES + SC_CERT_NONCE_BYTES)
```
Definition at line 29 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.1.14 SC_CERT_MARKER_BYTES

```
#define SC_CERT_MARKER_BYTES (1U)
```
Definition at line 25 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.1.15 SC_CERT_NONCE_BYTES

```
#define SC_CERT_NONCE_BYTES (8U)
```
Definition at line 26 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.1.16 SC_CERT_TOTAL_BYTES

```
#define SC_CERT_TOTAL_BYTES (74U)
```
Definition at line 28 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.1.17 SC_EC_COORD_BYTES

```
#define SC_EC_COORD_BYTES (32U)
```
Definition at line 21 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.29.1.18 SC_EC_PUBKEY_BYTES

```
#define SC_EC_PUBKEY_BYTES (64U)
```
Definition at line 22 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.29.1.19 SC_GET_CERT_RESP_BYTES

```
#define SC_GET_CERT_RESP_BYTES (148U)
```
Definition at line 34 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.1.20 SC_IV_BYTES

```
#define SC_IV_BYTES (16U)
```
Definition at line 38 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), and TEST_CASE().

5.29.1.21 SC_MUTUAL_AUTH_RESP_BYTES

```
#define SC_MUTUAL_AUTH_RESP_BYTES (66U)
```
Definition at line 36 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), and TEST_CASE().

5.29.1.22 SC_OPEN_SC_RESP_BYTES

```
#define SC_OPEN_SC_RESP_BYTES (34U)
```
Definition at line 35 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.1.23 SC_SALT_BYTES

```
#define SC_SALT_BYTES (32U)
```
Definition at line 37 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.29.1.24 SC_SELECT_RESP_BYTES

```
#define SC_SELECT_RESP_BYTES (26U)
```
Definition at line 33 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), and TEST_CASE().

5.29.1.25 SC_SHA512_OUT_BYTES

```
#define SC_SHA512_OUT_BYTES (64U)
```
Definition at line 39 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.1.26 SC_SW_BYTES

```
#define SC_SW_BYTES (2U)
```
Definition at line 32 of file test_cw_secure_channel.cpp.
Referenced by ReflectiveMockNfcTransport::sendAPDU().

5.29.2 Function Documentation

5.29.2.1 TEST_CASE() [1/12]

```
TEST_CASE (
            "aesCbcEncrypt/aesCbcDecrypt:  round-trip via reflective mock returns card
payload" ,
            "" [secure_channel])
```
Definition at line 639 of file test_cw_secure_channel.cpp.
References K_CARD_RESP_PLAINTEXT, K_TEST_KENC, K_TEST_KMAC, s_crypto, s_logger, s_platform, s_reflectiveTransport, SC_AES_KEY_BYTES, SC_CARD_RESP_PAYLOAD_BYTES, and SC_IV_BYTES.

5.29.2.2 TEST_CASE() [2/12]

```
TEST_CASE (
            "checkStatusWord:  response shorter than 2 bytes returns false" ,
            "" [secure_channel])
```
Definition at line 350 of file test_cw_secure_channel.cpp.
References s_crypto, s_logger, s_platform, and s_scriptedTransport.

5.29.2.3 TEST_CASE() [3/12]

```
TEST_CASE (
            "checkStatusWord:  SW 0x9000 returns true" ,
            "" [secure_channel])
```
Definition at line 324 of file test_cw_secure_channel.cpp.
References s_crypto, s_logger, s_platform, and s_scriptedTransport.

5.29.2.4 TEST_CASE() [4/12]

```
TEST_CASE (
            "checkStatusWord:  SW mismatch returns false" ,
            "" [secure_channel])
```
Definition at line 337 of file test_cw_secure_channel.cpp.
References s_crypto, s_logger, s_platform, and s_scriptedTransport.

5.29.2.5 TEST_CASE() [5/12]

```
TEST_CASE (
            "extractCardEphemeralKey:  extracts 64-byte key from synthetic certificate" ,
            "" [secure_channel])
```

Definition at line 365 of file test_cw_secure_channel.cpp.
References s_crypto, s_logger, s_platform, s_scriptedTransport, SC_CERT_KEY65_BYTES, SC_CERT_KEY_OFFSET, SC_CERT_MARKER_BYTES, SC_CERT_NONCE_BYTES, SC_CERT_TOTAL_BYTES, and SC_EC_PUBKEY_BYTES.

5.29.2.6 TEST_CASE() [6/12]

```
TEST_CASE (
            "extractCardEphemeralKey:  null cert pointer returns false" ,
            "" [secure_channel])
```

Definition at line 400 of file test_cw_secure_channel.cpp.
References s_crypto, s_logger, s_platform, s_scriptedTransport, and SC_EC_PUBKEY_BYTES.

5.29.2.7 TEST_CASE() [7/12]

```
TEST_CASE (
            "getCardCertificate:  extracts 146 certificate bytes from mock response" ,
            "" [secure_channel])
```

Definition at line 444 of file test_cw_secure_channel.cpp.
References s_crypto, s_logger, s_platform, s_scriptedTransport, and SC_GET_CERT_RESP_BYTES.

5.29.2.8 TEST_CASE() [8/12]

```
TEST_CASE (
            "key derivation:  ECDH + SHA-512 split yields distinct Kenc and Kmac" ,
            "" [secure_channel])
```

Definition at line 562 of file test_cw_secure_channel.cpp.
References s_crypto, s_logger, s_platform, s_scriptedTransport, SC_AES_KEY_BYTES, SC_EC_COORD_BYTES, SC_EC_PUBKEY_BYTES, SC_MUTUAL_AUTH_RESP_BYTES, SC_SALT_BYTES, and SC_SHA512_OUT_BYTES.

5.29.2.9 TEST_CASE() [9/12]

```
TEST_CASE (
            "mutuallyAuthenticate:  sets session IV to first 16 bytes of mock response" ,
            "" [secure_channel])
```

Definition at line 504 of file test_cw_secure_channel.cpp.
References K_CARD_EPHEMERAL_PUB, s_crypto, s_logger, s_platform, s_scriptedTransport, SC_AES_KEY_BYTES, SC_EC_COORD_BYTES, SC_EC_PUBKEY_BYTES, SC_IV_BYTES, SC_MUTUAL_AUTH_RESP_BYTES, and SC_SALT_BYTES.

5.29.2.10 TEST_CASE() [10/12]

```
TEST_CASE (
            "openSecureChannel:  extracts 32-byte salt from mock response" ,
            "" [secure_channel])
```

Definition at line 471 of file test_cw_secure_channel.cpp.
References s_crypto, s_logger, s_platform, s_scriptedTransport, SC_EC_COORD_BYTES, SC_EC_PUBKEY_BYTES, SC_OPEN_SC_RESP_BYTES, and SC_SALT_BYTES.

5.29.2.11 TEST_CASE() [11/12]

```
TEST_CASE (
            "selectApdu:  fails when transport returns error SW" ,
            "" [secure_channel])
```

Definition at line 429 of file test_cw_secure_channel.cpp.
References s_crypto, s_logger, s_platform, s_scriptedTransport, and SC_SELECT_RESP_BYTES.

5.29.2.12 TEST_CASE() [12/12]

```
TEST_CASE (
            "selectApdu:  succeeds when transport returns SW 0x9000" ,
            "" [secure_channel])
```

Definition at line 414 of file test_cw_secure_channel.cpp.
References s_crypto, s_logger, s_platform, s_scriptedTransport, and SC_SELECT_RESP_BYTES.

5.29.3 Variable Documentation

5.29.3.1 K_CARD_EPHEMERAL_PUB

```
const uint8_t K_CARD_EPHEMERAL_PUB[SC_EC_PUBKEY_BYTES]  [static]
```

Initial value:

```
= {

    0x60U, 0xfeU, 0xd4U, 0xbaU, 0x25U, 0x5aU, 0x9dU, 0x31U,
    0xc9U, 0x61U, 0xebU, 0x74U, 0xc6U, 0x35U, 0x6dU, 0x68U,
    0xc0U, 0x49U, 0xb8U, 0x92U, 0x3bU, 0x61U, 0xfaU, 0x6cU,
    0xe6U, 0x69U, 0x62U, 0x2eU, 0x60U, 0xf2U, 0x9fU, 0xb6U,

    0x79U, 0x03U, 0xfeU, 0x10U, 0x08U, 0xb8U, 0xbcU, 0x99U,
    0xa4U, 0x1aU, 0xe9U, 0xe9U, 0x56U, 0x28U, 0xbcU, 0x64U,
    0xf2U, 0xf1U, 0xb2U, 0x0cU, 0x2dU, 0x7eU, 0x9fU, 0x51U,
    0x77U, 0xa3U, 0xc2U, 0x94U, 0xd4U, 0x46U, 0x22U, 0x99U
}
```

Definition at line 59 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.3.2 K_CARD_RESP_PLAINTEXT

```
const uint8_t K_CARD_RESP_PLAINTEXT[SC_CARD_RESP_TOTAL_BYTES]  [static]
```

Initial value:

```
= {
    0xdeU, 0xadU, 0xbeU, 0xefU, 0x90U, 0x00U
}
```

Definition at line 88 of file test_cw_secure_channel.cpp.
Referenced by ReflectiveMockNfcTransport::sendAPDU(), and TEST_CASE().

5.29.3.3 K_TEST_KENC

```
const uint8_t K_TEST_KENC[SC_AES_KEY_BYTES]  [static]
```

Initial value:

```
= {
    0x60U, 0x3dU, 0xebU, 0x10U, 0x15U, 0xcaU, 0x71U, 0xbeU,
    0x2bU, 0x73U, 0xaeU, 0xf0U, 0x85U, 0x7dU, 0x77U, 0x81U,
    0x1fU, 0x35U, 0x2cU, 0x07U, 0x3bU, 0x61U, 0x08U, 0xd7U,
    0x2dU, 0x98U, 0x10U, 0xa3U, 0x09U, 0x14U, 0xdfU, 0xf4U
}
```

Definition at line 74 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.3.4 K_TEST_KMAC

```
const uint8_t K_TEST_KMAC[SC_AES_KEY_BYTES]  [static]
```

Initial value:

```
= {
    0x00U, 0x01U, 0x02U, 0x03U, 0x04U, 0x05U, 0x06U, 0x07U,
    0x08U, 0x09U, 0x0aU, 0x0bU, 0x0cU, 0x0dU, 0x0eU, 0x0fU,
    0x10U, 0x11U, 0x12U, 0x13U, 0x14U, 0x15U, 0x16U, 0x17U,
    0x18U, 0x19U, 0x1aU, 0x1bU, 0x1cU, 0x1dU, 0x1eU, 0x1fU
}
```

Definition at line 80 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.3.5 s_crypto

ESP32CryptoProvider s_crypto [static]
Definition at line 314 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.29.3.6 s_logger

MockLogger s_logger [static]
Definition at line 315 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.29.3.7 s_platform

MockPlatform s_platform [static]
Definition at line 316 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.29.3.8 s_reflectiveTransport

ReflectiveMockNfcTransport s_reflectiveTransport [static]
Definition at line 318 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE().

5.29.3.9 s_scriptedTransport

ScriptedMockNfcTransport s_scriptedTransport [static]
Definition at line 317 of file test_cw_secure_channel.cpp.
Referenced by TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), TEST_CASE(), and TEST_CASE().

5.30 test_cw_secure_channel.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #include "unity.h"
00007 #include "CW_SecureChannel.h"
00008 #include "CW_Defs.h"
00009 #include "CW_Utils.h"
00010 #include "CW_Platform.h"
00011 #include "esp32_crypto_provider.h"
00012 #include <string.h>
00013 #include <stdio.h>
00014
00015 /*****************************************************************
00016  * 1. Size / offset constants
00017  *****************************************************************/
00018
00019 #define SC_AES_BLOCK_BYTES          (16U)
00020 #define SC_AES_KEY_BYTES            (32U)
00021 #define SC_EC_COORD_BYTES           (32U)
00022 #define SC_EC_PUBKEY_BYTES          (64U)
00023
00024 /* Certificate layout: 'C'[1] + nonce[8] + 0x04[1] + X[32] + Y[32] = 74 bytes */
00025 #define SC_CERT_MARKER_BYTES         (1U)
00026 #define SC_CERT_NONCE_BYTES          (8U)
```

```
00027 #define SC_CERT_KEY65_BYTES         (65U)
00028 #define SC_CERT_TOTAL_BYTES         (74U)
00029 #define SC_CERT_KEY_OFFSET          (SC_CERT_MARKER_BYTES + SC_CERT_NONCE_BYTES)
00030
00031 /* APDU response sizes (data + 2-byte SW) */
00032 #define SC_SW_BYTES                  (2U)
00033 #define SC_SELECT_RESP_BYTES        (26U)
00034 #define SC_GET_CERT_RESP_BYTES     (148U)
00035 #define SC_OPEN_SC_RESP_BYTES       (34U)
00036 #define SC_MUTUAL_AUTH_RESP_BYTES   (66U)
00037 #define SC_SALT_BYTES               (32U)
00038 #define SC_IV_BYTES                 (16U)
00039 #define SC_SHA512_OUT_BYTES         (64U)
00040
00041 /* Secure messaging APDU layout (outgoing) */
00042 #define SC_APDU_HEADER_LEN           (4U)
00043 #define SC_APDU_LC_OFFSET            (4U)
00044 #define SC_APDU_MAC_OFFSET           (5U)   /* header[4] + Lc[1] */
00045 #define SC_APDU_MAC_BYTES           (16U)
00046
00047 /* Card response used in the AES-CBC round-trip test */
00048 #define SC_CARD_RESP_PAYLOAD_BYTES   (4U)
00049 #define SC_CARD_RESP_TOTAL_BYTES     (SC_CARD_RESP_PAYLOAD_BYTES + SC_SW_BYTES)
00050
00051 /* Mock scripting limits */
00052 #define MOCK_MAX_SCRIPTS             (8U)
00053 #define MOCK_MAX_RESP_BYTES        (255U)
00054 #define MOCK_UART_BAUD_RATE    (115200UL)
00055
00056 /* Known P-256 public key (NIST FIPS 186-4 test vector) used as the
00057  * card's ephemeral key in mock-transport tests.
00058  * 64 bytes = X[32] || Y[32] (no 0x04 prefix). */
00059 static const uint8_t K_CARD_EPHEMERAL_PUB[SC_EC_PUBKEY_BYTES] = {
00060     /* X */
00061     0x60U, 0xfeU, 0xd4U, 0xbaU, 0x25U, 0x5aU, 0x9dU, 0x31U,
00062     0xc9U, 0x61U, 0xebU, 0x74U, 0xc6U, 0x35U, 0x6dU, 0x68U,
00063     0xc0U, 0x49U, 0xb8U, 0x92U, 0x3bU, 0x61U, 0xfaU, 0x6cU,
00064     0xe6U, 0x69U, 0x62U, 0x2eU, 0x60U, 0xf2U, 0x9fU, 0xb6U,
00065     /* Y */
00066     0x79U, 0x03U, 0xfeU, 0x10U, 0x08U, 0xb8U, 0xbcU, 0x99U,
00067     0xa4U, 0x1aU, 0xe9U, 0xe9U, 0x56U, 0x28U, 0xbcU, 0x64U,
00068     0xf2U, 0xf1U, 0xb2U, 0x0cU, 0x2dU, 0x7eU, 0x9fU, 0x51U,
00069     0x77U, 0xa3U, 0xc2U, 0x94U, 0xd4U, 0x46U, 0x22U, 0x99U
00070 };
00071
00072 /* Known Kenc / Kmac for the AES-CBC round-trip test (AES-256 key from
00073  * NIST SP 800-38A, extended to 32 bytes for the second key). */
00074 static const uint8_t K_TEST_KENC[SC_AES_KEY_BYTES] = {
00075     0x60U, 0x3dU, 0xebU, 0x10U, 0x15U, 0xcaU, 0x71U, 0xbeU,
00076     0x2bU, 0x73U, 0xaeU, 0xf0U, 0x85U, 0x7dU, 0x77U, 0x81U,
00077     0x1fU, 0x35U, 0x2cU, 0x07U, 0x3bU, 0x61U, 0x08U, 0xd7U,
00078     0x2dU, 0x98U, 0x10U, 0xa3U, 0x09U, 0x14U, 0xdfU, 0xf4U
00079 };
00080 static const uint8_t K_TEST_KMAC[SC_AES_KEY_BYTES] = {
00081     0x00U, 0x01U, 0x02U, 0x03U, 0x04U, 0x05U, 0x06U, 0x07U,
00082     0x08U, 0x09U, 0x0aU, 0x0bU, 0x0cU, 0x0dU, 0x0eU, 0x0fU,
00083     0x10U, 0x11U, 0x12U, 0x13U, 0x14U, 0x15U, 0x16U, 0x17U,
00084     0x18U, 0x19U, 0x1aU, 0x1bU, 0x1cU, 0x1dU, 0x1eU, 0x1fU
00085 };
00086
00087 /* Card response payload returned by the reflective mock (4 bytes + SW). */
00088 static const uint8_t K_CARD_RESP_PLAINTEXT[SC_CARD_RESP_TOTAL_BYTES] = {
00089     0xdeU, 0xadU, 0xbeU, 0xefU, 0x90U, 0x00U
00090 };
00091
00092 /*****************************************************************
00093  * 2. MockLogger -- no-op implementation
00094  *****************************************************************/
00095
00096 class MockLogger : public CW_Logger {
00097 public:
00098     bool begin(unsigned long = MOCK_UART_BAUD_RATE) override {
00099         return true;
00100     }
00101     void print(const __FlashStringHelper*) override {
00102     }
00103     void print(const char*) override {
00104     }
00105     void print(char) override {
00106     }
00107     void print(uint8_t, int = DEC) override {
00108     }
00109     void print(uint16_t, int = DEC) override {
00110     }
00111     void print(uint32_t, int = DEC) override {
00112     }
00113     void print(int, int = DEC) override {
```

```
00114     }
00115     void println() override {
00116     }
00117     void println(const __FlashStringHelper*) override {
00118     }
00119     void println(const char*) override {
00120     }
00121     void println(char) override {
00122     }
00123     void println(uint8_t, int = DEC) override {
00124     }
00125     void println(uint16_t, int = DEC) override {
00126     }
00127     void println(uint32_t, int = DEC) override {
00128     }
00129     void println(int, int = DEC) override {
00130     }
00131     ~MockLogger() override {
00132     }
00133 };
00134
00135 /**************************************************************************
00136  * 3. MockPlatform -- no-op sleep_ms for tests
00137  **************************************************************************/
00138
00139 class MockPlatform : public CW_Platform {
00140 public:
00141     void sleep_ms(uint32_t /*ms*/) override {
00142     }
00143     ~MockPlatform() override {
00144     }
00145 };
00146
00147 /**************************************************************************
00148  * 4. ScriptedMockNfcTransport -- returns pre-loaded response buffers
00149  **************************************************************************/
00150
00151 struct MockScriptEntry {
00152     uint8_t data[MOCK_MAX_RESP_BYTES];
00153     uint8_t len;
00154     bool    succeed;
00155 };
00156
00157 class ScriptedMockNfcTransport : public CW_NfcTransport {
00158 public:
00159     MockScriptEntry scripts[MOCK_MAX_SCRIPTS]{};
00160     uint8_t         scriptCount = 0U;
00161     uint8_t         callIdx     = 0U;
00162
00163     void reset() {
00164         scriptCount = 0U;
00165         callIdx     = 0U;
00166         (void)memset(scripts, 0, sizeof(scripts));
00167     }
00168
00169     void addScript(const uint8_t* data, uint8_t len, bool succeed = true) {
00170         if (scriptCount < MOCK_MAX_SCRIPTS) {
00171             (void)memcpy(scripts[scriptCount].data, data, static_cast<size_t>(len));
00172             scripts[scriptCount].len     = len;
00173             scripts[scriptCount].succeed = succeed;
00174             scriptCount++;
00175         }
00176     }
00177
00178     bool begin() override {
00179         return true;
00180     }
00181     bool inListPassiveTarget() override {
00182         return true;
00183     }
00184     void resetReader() override {
00185     }
00186     bool printFirmwareVersion() override {
00187         return true;
00188     }
00189
00190     bool sendAPDU(const uint8_t* apdu, uint8_t apduLen,
00191                   uint8_t* response, uint8_t& responseLen) override {
00192         bool result = false;
00193         (void)apdu;
00194         (void)apduLen;
00195
00196         if (callIdx < scriptCount) {
00197             const MockScriptEntry& e = scripts[callIdx];
00198             callIdx++;
00199             if (e.succeed) {
00200                 (void)memcpy(response, e.data, static_cast<size_t>(e.len));
```

```
00201                 responseLen = e.len;
00202                 result = true;
00203             }
00204         }
00205         return result;
00206     }
00207
00208     ~ScriptedMockNfcTransport() override {
00209     }
00210 };
00211
00212 /**************************************************************
00213  * 4. ReflectiveMockNfcTransport -- computes a valid encrypted response
00214  *    for the AES-CBC-MAC round-trip test.
00215  *
00216  *    Expected incoming APDU layout (from CW_SecureChannel::aesCbcEncrypt):
00217  *      header[4] | Lc[1] | MAC[16] | ciphertext[N]
00218  *
00219  *    The mock extracts the sent MAC, encrypts K_CARD_RESP_PLAINTEXT using
00220  *    the session Kenc with that MAC as IV (matching what aesCbcDecrypt
00221  *    expects), and wraps it in the response MAC.
00222  **************************************************************/
00223
00224 class ReflectiveMockNfcTransport : public CW_NfcTransport {
00225 public:
00226     const CW_SecureSession* session = nullptr;
00227     CW_CryptoProvider*      crypto  = nullptr;
00228
00229     bool begin() override {
00230         return true;
00231     }
00232     bool inListPassiveTarget() override {
00233         return true;
00234     }
00235     void resetReader() override {
00236     }
00237     bool printFirmwareVersion() override {
00238         return true;
00239     }
00240
00241     bool sendAPDU(const uint8_t* apdu, uint8_t apduLen,
00242                   uint8_t* response, uint8_t& responseLen) override {
00243         bool result = false;
00244
00245         if ((apdu != NULL) &&
00246             (apduLen > static_cast<uint8_t>(SC_APDU_MAC_OFFSET + SC_APDU_MAC_BYTES))) {
00247             /* Step 1: extract sentMAC -- used as IV when the channel decrypts the response */
00248             uint8_t sentMacIv[SC_AES_BLOCK_BYTES] = { 0U };
00249             (void)memcpy(sentMacIv, apdu + SC_APDU_MAC_OFFSET, SC_AES_BLOCK_BYTES);
00250
00251             /* Step 2: encrypt K_CARD_RESP_PLAINTEXT using Kenc, sentMAC as IV, bit-padding */
00252             uint8_t cipherResp[SC_AES_BLOCK_BYTES * 2U] = { 0U };
00253             uint8_t encIv[SC_AES_BLOCK_BYTES] = { 0U };
00254             (void)memcpy(encIv, sentMacIv, SC_AES_BLOCK_BYTES);
00255
00256             uint16_t cipherRespLen = crypto->aesCbcEncrypt(
00257                 K_CARD_RESP_PLAINTEXT,
00258                 static_cast<uint16_t>(sizeof(K_CARD_RESP_PLAINTEXT)),
00259                 cipherResp,
00260                 session->aesKey,
00261                 static_cast<uint8_t>(sizeof(session->aesKey)),
00262                 encIv,
00263                 true);
00264
00265             /* Step 3: build the response MAC input:
00266              *   [totalDataLen as 1st byte of a 16-byte zero block] || [cipherResp] */
00267             uint8_t totalDataLen = static_cast<uint8_t>(SC_AES_BLOCK_BYTES + cipherRespLen);
00268             uint8_t macInput[SC_AES_BLOCK_BYTES + SC_AES_BLOCK_BYTES * 2U] = { 0U };
00269             macInput[0U] = totalDataLen;
00270             (void)memcpy(macInput + SC_AES_BLOCK_BYTES, cipherResp,
00271                          static_cast<size_t>(cipherRespLen));
00272
00273             uint16_t macInputLen = static_cast<uint16_t>(SC_AES_BLOCK_BYTES) + cipherRespLen;
00274
00275             uint8_t macZeroIv[SC_AES_BLOCK_BYTES] = { 0U };
00276             uint8_t macOut[SC_AES_BLOCK_BYTES + SC_AES_BLOCK_BYTES * 2U] = { 0U };
00277             uint16_t macOutLen = crypto->aesCbcEncrypt(
00278                 macInput,
00279                 macInputLen,
00280                 macOut,
00281                 session->macKey,
00282                 static_cast<uint8_t>(sizeof(session->macKey)),
00283                 macZeroIv,
00284                 false);
00285
00286             /* Step 4: responseMac = last 16 bytes of macOut */
00287             const uint8_t* responseMac =
```

```
00288                 macOut + macOutLen - static_cast<uint16_t>(SC_AES_BLOCK_BYTES);
00289
00290             /* Step 5: assemble response = responseMac[16] || cipherResp[N] || SW[2] */
00291             uint8_t totalRespLen = static_cast<uint8_t>(
00292                 static_cast<uint16_t>(SC_AES_BLOCK_BYTES) + cipherRespLen + SC_SW_BYTES);
00293
00294             (void)memcpy(response, responseMac, SC_AES_BLOCK_BYTES);
00295             (void)memcpy(response + SC_AES_BLOCK_BYTES, cipherResp,
00296                          static_cast<size_t>(cipherRespLen));
00297             response[SC_AES_BLOCK_BYTES + cipherRespLen]             = 0x90U;
00298             response[SC_AES_BLOCK_BYTES + cipherRespLen + 1U] = 0x00U;
00299             responseLen = totalRespLen;
00300             result = true;
00301         }
00302
00303         return result;
00304     }
00305
00306     ~ReflectiveMockNfcTransport() override {
00307     }
00308 };
00309
00310 /****************************************************************
00311  * 5. Static instances shared across tests
00312  ****************************************************************/
00313
00314 static ESP32CryptoProvider        s_crypto;
00315 static MockLogger                 s_logger;
00316 static MockPlatform               s_platform;
00317 static ScriptedMockNfcTransport   s_scriptedTransport;
00318 static ReflectiveMockNfcTransport s_reflectiveTransport;
00319
00320 /****************************************************************
00321  * 6. checkStatusWord tests  (section numbering kept for diff clarity)
00322  ****************************************************************/
00323
00324 TEST_CASE("checkStatusWord: SW 0x9000 returns true", "[secure_channel]")
00325 {
00326     static const uint8_t resp[] = {
00327         0x01U, 0x02U, 0x03U, 0x04U, 0x90U, 0x00U
00328     };
00329     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00330
00331     bool ok = channel.checkStatusWord(resp, static_cast<uint8_t>(sizeof(resp)),
00332                                       0x90U, 0x00U);
00333
00334     TEST_ASSERT_TRUE(ok);
00335 }
00336
00337 TEST_CASE("checkStatusWord: SW mismatch returns false", "[secure_channel]")
00338 {
00339     static const uint8_t resp[] = {
00340         0x01U, 0x02U, 0x6aU, 0x82U
00341     };
00342     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00343
00344     bool ok = channel.checkStatusWord(resp, static_cast<uint8_t>(sizeof(resp)),
00345                                       0x90U, 0x00U);
00346
00347     TEST_ASSERT_FALSE(ok);
00348 }
00349
00350 TEST_CASE("checkStatusWord: response shorter than 2 bytes returns false", "[secure_channel]")
00351 {
00352     static const uint8_t resp[] = { 0x90U };
00353     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00354
00355     bool ok = channel.checkStatusWord(resp, static_cast<uint8_t>(sizeof(resp)),
00356                                       0x90U, 0x00U);
00357
00358     TEST_ASSERT_FALSE(ok);
00359 }
00360
00361 /****************************************************************
00362  * 7. extractCardEphemeralKey tests
00363  ****************************************************************/
00364
00365 TEST_CASE("extractCardEphemeralKey: extracts 64-byte key from synthetic certificate",
00366           "[secure_channel]")
00367 {
00368     /* Build a 74-byte synthetic certificate:
00369      *   cert[0]     = 'C' marker
00370      *   cert[1..8] = nonce (arbitrary)
00371      *   cert[9]     = 0x04 (uncompressed prefix -- dropped by the extractor)
00372      *   cert[10..41] = X coordinate (known pattern)
00373      *   cert[42..73] = Y coordinate (known pattern) */
00374     uint8_t cert[SC_CERT_TOTAL_BYTES] = { 0U };
```

```
00375     cert[0U] = 0x43U;  /* 'C' */
00376     for (uint8_t i = 0U; i < static_cast<uint8_t>(SC_CERT_NONCE_BYTES); i++) {
00377         cert[static_cast<size_t>(SC_CERT_MARKER_BYTES) + i] = static_cast<uint8_t>(i + 1U);
00378     }
00379     cert[SC_CERT_KEY_OFFSET] = 0x04U;  /* uncompressed prefix */
00380     for (uint8_t i = 1U; i < static_cast<uint8_t>(SC_CERT_KEY65_BYTES); i++) {
00381         cert[static_cast<size_t>(SC_CERT_KEY_OFFSET) + i] = static_cast<uint8_t>(0xA0U + i);
00382     }
00383
00384     uint8_t pubKey[SC_EC_PUBKEY_BYTES]    = { 0U };
00385     uint8_t fullKey65[SC_CERT_KEY65_BYTES] = { 0U };
00386
00387     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00388     bool ok = channel.extractCardEphemeralKey(cert, pubKey, fullKey65);
00389
00390     TEST_ASSERT_TRUE(ok);
00391
00392     /* fullKey65[0] must be the 0x04 prefix */
00393     TEST_ASSERT_EQUAL_HEX8(0x04U, fullKey65[0U]);
00394
00395     /* pubKey = fullKey65[1..64] = cert[10..73] */
00396     TEST_ASSERT_EQUAL_HEX8_ARRAY(cert + SC_CERT_KEY_OFFSET + 1U, pubKey,
00397                                  SC_EC_PUBKEY_BYTES);
00398 }
00399
00400 TEST_CASE("extractCardEphemeralKey: null cert pointer returns false", "[secure_channel]")
00401 {
00402     uint8_t pubKey[SC_EC_PUBKEY_BYTES] = { 0U };
00403
00404     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00405     bool ok = channel.extractCardEphemeralKey(NULL, pubKey, NULL);
00406
00407     TEST_ASSERT_FALSE(ok);
00408 }
00409
00410 /****************************************************************
00411  * 8. Protocol-flow tests with ScriptedMockNfcTransport
00412  ****************************************************************/
00413
00414 TEST_CASE("selectApdu: succeeds when transport returns SW 0x9000", "[secure_channel]")
00415 {
00416     uint8_t resp[SC_SELECT_RESP_BYTES] = { 0U };
00417     resp[SC_SELECT_RESP_BYTES - 2U] = 0x90U;
00418     resp[SC_SELECT_RESP_BYTES - 1U] = 0x00U;
00419
00420     s_scriptedTransport.reset();
00421     s_scriptedTransport.addScript(resp, static_cast<uint8_t>(sizeof(resp)));
00422
00423     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00424     bool ok = channel.selectApdu();
00425
00426     TEST_ASSERT_TRUE(ok);
00427 }
00428
00429 TEST_CASE("selectApdu: fails when transport returns error SW", "[secure_channel]")
00430 {
00431     uint8_t resp[SC_SELECT_RESP_BYTES] = { 0U };
00432     resp[SC_SELECT_RESP_BYTES - 2U] = 0x6aU;
00433     resp[SC_SELECT_RESP_BYTES - 1U] = 0x82U;
00434
00435     s_scriptedTransport.reset();
00436     s_scriptedTransport.addScript(resp, static_cast<uint8_t>(sizeof(resp)));
00437
00438     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00439     bool ok = channel.selectApdu();
00440
00441     TEST_ASSERT_FALSE(ok);
00442 }
00443
00444 TEST_CASE("getCardCertificate: extracts 146 certificate bytes from mock response",
00445           "[secure_channel]")
00446 {
00447     /* Build a 148-byte scripted response:
00448      *   bytes[0..145] = certificate data
00449      *   bytes[146..147] = SW 0x90 0x00 */
00450     uint8_t resp[SC_GET_CERT_RESP_BYTES] = { 0U };
00451     for (uint8_t i = 0U; i < static_cast<uint8_t>(SC_GET_CERT_RESP_BYTES - 2U); i++) {
00452         resp[i] = static_cast<uint8_t>(i + 1U);
00453     }
00454     resp[SC_GET_CERT_RESP_BYTES - 2U] = 0x90U;
00455     resp[SC_GET_CERT_RESP_BYTES - 1U] = 0x00U;
00456
00457     s_scriptedTransport.reset();
00458     s_scriptedTransport.addScript(resp, static_cast<uint8_t>(sizeof(resp)));
00459
00460     uint8_t certBuf[SC_GET_CERT_RESP_BYTES] = { 0U };
00461     uint8_t certLen = 0U;
```

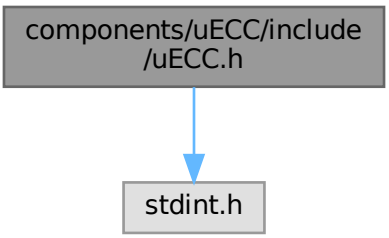
```
00462
00463     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00464     bool ok = channel.getCardCertificate(certBuf, certLen);
00465
00466     TEST_ASSERT_TRUE(ok);
00467     TEST_ASSERT_EQUAL_UINT8(static_cast<uint8_t>(SC_GET_CERT_RESP_BYTES - 2U), certLen);
00468     TEST_ASSERT_EQUAL_HEX8_ARRAY(resp, certBuf, certLen);
00469 }
00470
00471 TEST_CASE("openSecureChannel: extracts 32-byte salt from mock response", "[secure_channel]")
00472 {
00473     /* Build 34-byte scripted response: 32-byte salt + SW */
00474     uint8_t resp[SC_OPEN_SC_RESP_BYTES] = { 0U };
00475     for (uint8_t i = 0U; i < static_cast<uint8_t>(SC_SALT_BYTES); i++) {
00476         resp[i] = static_cast<uint8_t>(0xA0U + i);
00477     }
00478     resp[SC_OPEN_SC_RESP_BYTES - 2U] = 0x90U;
00479     resp[SC_OPEN_SC_RESP_BYTES - 1U] = 0x00U;
00480
00481     s_scriptedTransport.reset();
00482     s_scriptedTransport.addScript(resp, static_cast<uint8_t>(sizeof(resp)));
00483
00484     uint8_t salt[SC_SALT_BYTES]              = { 0U };
00485     uint8_t clientPub[SC_EC_PUBKEY_BYTES]    = { 0U };
00486     uint8_t clientPriv[SC_EC_COORD_BYTES]    = { 0U };
00487     CW_Curve curve = CW_CURVE_SECP256R1;
00488
00489     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00490     bool ok = channel.openSecureChannel(salt, clientPub, clientPriv, curve);
00491
00492     TEST_ASSERT_TRUE(ok);
00493
00494     /* Salt must match the first 32 bytes of the scripted response */
00495     TEST_ASSERT_EQUAL_HEX8_ARRAY(resp, salt, SC_SALT_BYTES);
00496
00497     /* Client keypair must have been generated (non-zero) */
00498     const uint8_t zeroPub[SC_EC_PUBKEY_BYTES] = { 0U };
00499     const uint8_t zeroPriv[SC_EC_COORD_BYTES] = { 0U };
00500     TEST_ASSERT_FALSE(CW_Utils::secure_compare(clientPub,  zeroPub,  SC_EC_PUBKEY_BYTES));
00501     TEST_ASSERT_FALSE(CW_Utils::secure_compare(clientPriv, zeroPriv, SC_EC_COORD_BYTES));
00502 }
00503
00504 TEST_CASE("mutuallyAuthenticate: sets session IV to first 16 bytes of mock response",
00505           "[secure_channel]")
00506 {
00507     /* Scripted 66-byte response for MUTUALLY AUTHENTICATE:
00508      *   bytes[0..15]  = rolling IV (known pattern)
00509      *   bytes[16..63] = dummy encrypted data
00510      *   bytes[64..65] = SW 0x90 0x00 */
00511     uint8_t resp[SC_MUTUAL_AUTH_RESP_BYTES] = { 0U };
00512     for (uint8_t i = 0U; i < static_cast<uint8_t>(SC_IV_BYTES); i++) {
00513         resp[i] = static_cast<uint8_t>(0xC0U + i);
00514     }
00515     resp[SC_MUTUAL_AUTH_RESP_BYTES - 2U] = 0x90U;
00516     resp[SC_MUTUAL_AUTH_RESP_BYTES - 1U] = 0x00U;
00517
00518     s_scriptedTransport.reset();
00519     s_scriptedTransport.addScript(resp, static_cast<uint8_t>(sizeof(resp)));
00520
00521     /* Use the known P-256 public key as the card's ephemeral key.
00522      * Any valid point on secp256r1 works here -- ECDH must not fail. */
00523     uint8_t cardEphemeralPub[SC_EC_PUBKEY_BYTES] = { 0U };
00524     (void)memcpy(cardEphemeralPub, K_CARD_EPHEMERAL_PUB, SC_EC_PUBKEY_BYTES);
00525
00526     uint8_t clientPub[SC_EC_PUBKEY_BYTES]  = { 0U };
00527     uint8_t clientPriv[SC_EC_COORD_BYTES]  = { 0U };
00528     const uint8_t salt[SC_SALT_BYTES]      = { 0U };
00529     CW_Curve curve = CW_CURVE_SECP256R1;
00530
00531     bool keyOk = s_crypto.makeKey(clientPub, clientPriv, curve);
00532     TEST_ASSERT_TRUE(keyOk);
00533
00534     CW_SecureSession session;
00535     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00536     bool ok = channel.mutuallyAuthenticate(session, salt, clientPub, clientPriv,
00537                                            curve, cardEphemeralPub);
00538
00539     TEST_ASSERT_TRUE(ok);
00540
00541     /* Session IV must be the first 16 bytes of the scripted response */
00542     TEST_ASSERT_EQUAL_HEX8_ARRAY(resp, session.iv, SC_IV_BYTES);
00543
00544     /* Session keys must be non-zero (ECDH + SHA-512 derivation ran) */
00545     const uint8_t zeroKey[SC_AES_KEY_BYTES] = { 0U };
00546     TEST_ASSERT_FALSE(CW_Utils::secure_compare(session.aesKey, zeroKey, SC_AES_KEY_BYTES));
00547     TEST_ASSERT_FALSE(CW_Utils::secure_compare(session.macKey, zeroKey, SC_AES_KEY_BYTES));
00548 }
```

```
00549
00550 /**********************************************************************
00551  * 9. Key derivation correctness test
00552  *
00553  *    The secure channel derives Kenc and Kmac as:
00554  *      sha512( ecdh_secret || "Cryptnox Basic CommonPairingData" || salt )
00555  *    Kenc = output[0..31], Kmac = output[32..63].
00556  *
00557  *    This test verifies the derivation is consistent by computing it
00558  *    independently and comparing with what SHA-512 over the same input
00559  *    produces directly through the crypto provider.
00560  **********************************************************************/
00561
00562 TEST_CASE("key derivation: ECDH + SHA-512 split yields distinct Kenc and Kmac",
00563           "[secure_channel]")
00564 {
00565     CW_Curve curve = CW_CURVE_SECP256R1;
00566
00567     /* Generate two ephemeral keypairs */
00568     uint8_t pubA[SC_EC_PUBKEY_BYTES]  = { 0U };
00569     uint8_t privA[SC_EC_COORD_BYTES]  = { 0U };
00570     uint8_t pubB[SC_EC_PUBKEY_BYTES]  = { 0U };
00571     uint8_t privB[SC_EC_COORD_BYTES]  = { 0U };
00572
00573     bool okA = s_crypto.makeKey(pubA, privA, curve);
00574     bool okB = s_crypto.makeKey(pubB, privB, curve);
00575     TEST_ASSERT_TRUE(okA);
00576     TEST_ASSERT_TRUE(okB);
00577
00578     /* ECDH: both sides must yield the same shared secret */
00579     uint8_t secretAB[SC_EC_COORD_BYTES] = { 0U };
00580     uint8_t secretBA[SC_EC_COORD_BYTES] = { 0U };
00581     bool ecdhAB = s_crypto.ecdh(pubB, privA, secretAB, curve);
00582     bool ecdhBA = s_crypto.ecdh(pubA, privB, secretBA, curve);
00583     TEST_ASSERT_TRUE(ecdhAB);
00584     TEST_ASSERT_TRUE(ecdhBA);
00585     TEST_ASSERT_EQUAL_HEX8_ARRAY(secretAB, secretBA, SC_EC_COORD_BYTES);
00586
00587     /* Manually derive Kenc/Kmac with a known salt */
00588     static const uint8_t SALT[SC_SALT_BYTES] = {
00589         0x11U, 0x22U, 0x33U, 0x44U, 0x55U, 0x66U, 0x77U, 0x88U,
00590         0x99U, 0xaaU, 0xbbU, 0xccU, 0xddU, 0xeeU, 0xffU, 0x00U,
00591         0x01U, 0x02U, 0x03U, 0x04U, 0x05U, 0x06U, 0x07U, 0x08U,
00592         0x09U, 0x0aU, 0x0bU, 0x0cU, 0x0dU, 0x0eU, 0x0fU, 0x10U
00593     };
00594
00595     uint8_t concat[SC_EC_COORD_BYTES + CW_PAIRING_DATA_BYTES + SC_SALT_BYTES] = { 0U };
00596     (void)memcpy(concat, secretAB, SC_EC_COORD_BYTES);
00597     (void)memcpy(concat + SC_EC_COORD_BYTES, CW_PAIRING_DATA, CW_PAIRING_DATA_BYTES);
00598     (void)memcpy(concat + SC_EC_COORD_BYTES + CW_PAIRING_DATA_BYTES, SALT, SC_SALT_BYTES);
00599
00600     uint8_t sha512Out[SC_SHA512_OUT_BYTES] = { 0U };
00601     s_crypto.sha512(concat, sizeof(concat), sha512Out);
00602
00603     /* Kenc = sha512[0..31], Kmac = sha512[32..63] -- they must differ */
00604     uint8_t* derivedKenc = sha512Out;
00605     uint8_t* derivedKmac = sha512Out + SC_AES_KEY_BYTES;
00606
00607     TEST_ASSERT_FALSE(CW_Utils::secure_compare(derivedKenc, derivedKmac, SC_AES_KEY_BYTES));
00608
00609     /* Cross-check: running mutuallyAuthenticate with these inputs must set
00610      * the same Kenc/Kmac in the session (one scripted APDU response). */
00611     uint8_t mutualResp[SC_MUTUAL_AUTH_RESP_BYTES] = { 0U };
00612     mutualResp[SC_MUTUAL_AUTH_RESP_BYTES - 2U] = 0x90U;
00613     mutualResp[SC_MUTUAL_AUTH_RESP_BYTES - 1U] = 0x00U;
00614
00615     s_scriptedTransport.reset();
00616     s_scriptedTransport.addScript(mutualResp,
00617                                   static_cast<uint8_t>(sizeof(mutualResp)));
00618
00619     CW_SecureSession session;
00620     CW_SecureChannel channel(s_scriptedTransport, s_logger, s_crypto, s_platform);
00621
00622     /* publicB acts as the "card" ephemeral key; privA is the client key.
00623      * The ECDH inside mutuallyAuthenticate will compute secretAB. */
00624     bool ok = channel.mutuallyAuthenticate(session, SALT, pubA, privA, curve, pubB);
00625     TEST_ASSERT_TRUE(ok);
00626
00627     TEST_ASSERT_EQUAL_HEX8_ARRAY(derivedKenc, session.aesKey, SC_AES_KEY_BYTES);
00628     TEST_ASSERT_EQUAL_HEX8_ARRAY(derivedKmac, session.macKey, SC_AES_KEY_BYTES);
00629 }
00630
00631 /**********************************************************************
00632  * 10. AES-CBC-MAC round-trip: aesCbcEncrypt + aesCbcDecrypt
00633  *
00634  *     Uses a manually set session (known Kenc/Kmac) and the
00635  *     ReflectiveMockNfcTransport which computes a valid card response.
```

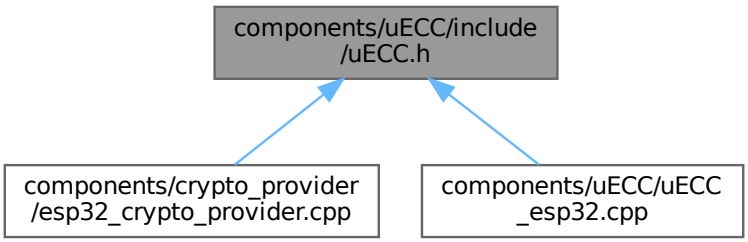
```
00636  *     Verifies the decrypted payload matches K_CARD_RESP_PLAINTEXT[0..3].
00637  ****************************************************************/
00638
00639 TEST_CASE("aesCbcEncrypt/aesCbcDecrypt: round-trip via reflective mock returns card payload",
00640           "[secure_channel]")
00641 {
00642     /* Set up a session with known keys and a zero IV */
00643     CW_SecureSession session;
00644     (void)memcpy(session.aesKey, K_TEST_KENC, SC_AES_KEY_BYTES);
00645     (void)memcpy(session.macKey, K_TEST_KMAC, SC_AES_KEY_BYTES);
00646     (void)memset(session.iv, 0x00U, SC_IV_BYTES);
00647
00648     /* Wire the reflective mock to this session */
00649     s_reflectiveTransport.session = &session;
00650     s_reflectiveTransport.crypto  = &s_crypto;
00651
00652     CW_SecureChannel channel(s_reflectiveTransport, s_logger, s_crypto, s_platform);
00653
00654     /* Arbitrary plaintext command payload */
00655     static const uint8_t plaintext[]   = { 0xAAU, 0xBBU, 0xCCU };
00656     static const uint8_t apduHeader[]  = { 0x80U, 0x01U, 0x00U, 0x00U };
00657
00658     uint8_t  decryptedOut[32U] = { 0U };
00659     uint16_t decryptedLen      = 0U;
00660
00661     bool ok = channel.aesCbcEncrypt(
00662         session,
00663         apduHeader,
00664         static_cast<uint16_t>(sizeof(apduHeader)),
00665         plaintext,
00666         static_cast<uint16_t>(sizeof(plaintext)),
00667         decryptedOut,
00668         &decryptedLen);
00669
00670     TEST_ASSERT_TRUE(ok);
00671
00672     /* decryptedLen = card response payload without the two SW bytes */
00673     TEST_ASSERT_EQUAL_UINT16(static_cast<uint16_t>(SC_CARD_RESP_PAYLOAD_BYTES),
00674                              decryptedLen);
00675
00676     /* Decrypted payload must match K_CARD_RESP_PLAINTEXT (excl. SW) */
00677     TEST_ASSERT_EQUAL_HEX8_ARRAY(K_CARD_RESP_PLAINTEXT, decryptedOut,
00678                                  SC_CARD_RESP_PAYLOAD_BYTES);
00679 }
```

5.31 components/uECC/include/uECC.h File Reference

`#include <stdint.h>`
Include dependency graph for uECC.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef struct uECC_Curve_t uECC_Curve_t
- typedef int(∗ uECC_RNG_Function) (uint8_t ∗dest, unsigned size)

Functions

- const uECC_Curve_t ∗ uECC_secp256r1 (void)

  Return the static secp256r1 curve descriptor.
- const uECC_Curve_t ∗ uECC_secp256k1 (void)

  Return the static secp256k1 curve descriptor.
- void uECC_set_rng (uECC_RNG_Function rng_function)

  No-op: ESP32 hardware RNG is used internally; no external callback needed.
- int uECC_make_key (uint8_t ∗public_key, uint8_t ∗private_key, const uECC_Curve_t ∗curve)

  Generate an ECC key pair using mbedTLS and the ESP32 hardware RNG.
- int uECC_shared_secret (const uint8_t ∗public_key, const uint8_t ∗private_key, uint8_t ∗secret, const uECC_Curve_t ∗curve)

  Compute ECDH shared secret (X-coordinate of privKey ∗ pubKey).
- int uECC_verify (const uint8_t ∗public_key, const uint8_t ∗hash, unsigned hash_size, const uint8_t ∗signature, const uECC_Curve_t ∗curve)

  Verify an ECDSA signature (raw 64-byte r||s) against a hash.

5.31.1 Typedef Documentation

5.31.1.1 uECC_Curve_t

```
typedef struct uECC_Curve_t uECC_Curve_t
```

Definition at line 22 of file uECC.h.

5.31.1.2 uECC_RNG_Function

```
typedef int(* uECC_RNG_Function) (uint8_t *dest, unsigned size)
```

Definition at line 32 of file uECC.h.

5.31.2 Function Documentation

5.31.2.1 uECC_make_key()

```
int uECC_make_key (
            uint8_t * public_key,
            uint8_t * private_key,
            const uECC_Curve_t * curve)
```

Generate an ECC key pair using mbedTLS and the ESP32 hardware RNG.
Definition at line 92 of file uECC_esp32.cpp.
References COORD_SIZE_BYTES, COORD_X_OFFSET, ECC_XY_KEY_SIZE, esp32_mbedtls_rng(), uECC_Curve_t::grp_id, MBEDTLS_OK, UECC_FAILURE, UECC_SUCCESS, and UNCOMPRESSED_PUB_SIZE.
Referenced by ESP32CryptoProvider::makeKey().

5.31.2.2 uECC_secp256k1()

```
const uECC_Curve_t * uECC_secp256k1 (
            void )
```

Return the static secp256k1 curve descriptor.
Definition at line 75 of file uECC_esp32.cpp.
References s_secp256k1.
Referenced by toCurve().

5.31.2.3 uECC_secp256r1()

```
const uECC_Curve_t * uECC_secp256r1 (
            void )
```

Return the static secp256r1 curve descriptor.
Definition at line 70 of file uECC_esp32.cpp.
References s_secp256r1.
Referenced by toCurve().

5.31.2.4 uECC_set_rng()

```
void uECC_set_rng (
            uECC_RNG_Function rng_function)
```

No-op: ESP32 hardware RNG is used internally; no external callback needed.
Definition at line 80 of file uECC_esp32.cpp.
References UECC_LOG_TAG.

5.31.2.5 uECC_shared_secret()

```
int uECC_shared_secret (
            const uint8_t * public_key,
            const uint8_t * private_key,
            uint8_t * secret,
            const uECC_Curve_t * curve)
```

Compute ECDH shared secret (X-coordinate of privKey * pubKey).
Definition at line 144 of file uECC_esp32.cpp.
References COORD_SIZE_BYTES, COORD_X_OFFSET, ECC_XY_KEY_SIZE, esp32_mbedtls_rng(), uECC_Curve_t::grp_id, MBEDTLS_OK, POINT_PREFIX_OFFSET, UECC_FAILURE, UECC_SUCCESS, UNCOMPRESSED_PREFIX, and UNCOMPRESSED_PUB_SIZE.
Referenced by ESP32CryptoProvider::ecdh().

5.31.2.6 uECC_verify()

```
int uECC_verify (
            const uint8_t * public_key,
            const uint8_t * hash,
            unsigned hash_size,
            const uint8_t * signature,
            const uECC_Curve_t * curve)
```

Verify an ECDSA signature (raw 64-byte r||s) against a hash.
Definition at line 210 of file uECC_esp32.cpp.
References COORD_SIZE_BYTES, COORD_X_OFFSET, ECC_XY_KEY_SIZE, uECC_Curve_t::grp_id, MBEDTLS_OK, POINT_PREFIX_OFFSET, RAW_SIG_R_OFFSET, RAW_SIG_S_OFFSET, UECC_FAILURE, UECC_SUCCESS, UNCOMPRESSED_PREFIX, and UNCOMPRESSED_PUB_SIZE.
Referenced by ESP32CryptoProvider::ecdsaVerify().

5.32 uECC.h

Go to the documentation of this file.

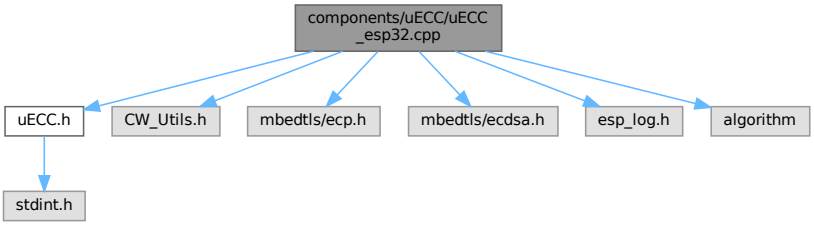
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #ifndef UECC_H
00007 #define UECC_H
00008
00009 /*
00010  * uECC compatibility shim for ESP32.
00011  * Exposes the micro-ecc API surface required by cryptnox-sdk-cpp,
00012  * implemented internally via mbedTLS and the ESP32 hardware RNG.
00013  */
00014
00015 #ifdef __cplusplus
00016 extern "C" {
00017 #endif
00018
00019 #include <stdint.h>
00020
00021 /* Opaque curve descriptor -- definition lives in uECC_esp32.cpp. */
00022 typedef struct uECC_Curve_t uECC_Curve_t;
00023
00024 /* Return curve descriptors for the two Cryptnox curves. */
00025 const uECC_Curve_t* uECC_secp256r1(void);
00026 const uECC_Curve_t* uECC_secp256k1(void);
00027
00028 /*
00029  * RNG callback type (micro-ecc compatible).
00030  * Returns 1 on success, 0 on failure.
00031  */
00032 typedef int (*uECC_RNG_Function)(uint8_t *dest, unsigned size);
00033
00034 /*
00035  * Register an external RNG callback.
00036  * On ESP32 this is a no-op -- the hardware RNG is used via mbedTLS directly.
00037  */
00038 void uECC_set_rng(uECC_RNG_Function rng_function);
00039
00040 /*
00041  * Generate an EC key pair.
00042  *   public_key  [out] 64 bytes: X || Y (uncompressed, no 0x04 prefix).
00043  *   private_key [out] 32 bytes.
00044  * Returns 1 on success, 0 on failure.
00045  */
00046 int uECC_make_key(uint8_t *public_key, uint8_t *private_key,
00047                   const uECC_Curve_t *curve);
00048
00049 /*
00050  * Compute an ECDH shared secret.
00051  *   public_key  [in]  64-byte remote public key (X || Y, no 0x04 prefix).
00052  *   private_key [in]  32-byte local private key.
00053  *   secret      [out] 32-byte shared secret (X coordinate of d*Q).
00054  * Returns 1 on success, 0 on failure.
00055  */
00056 int uECC_shared_secret(const uint8_t *public_key, const uint8_t *private_key,
00057                        uint8_t *secret, const uECC_Curve_t *curve);
00058
00059 /*
00060  * Verify an ECDSA signature.
00061  *   public_key [in] 64-byte public key (X || Y, no 0x04 prefix).
00062  *   hash       [in] Message hash bytes.
00063  *   hash_size  [in] Hash length in bytes (32 for SHA-256).
00064  *   signature  [in] 64-byte raw signature: r[32] || s[32].
00065  * Returns 1 if valid, 0 if invalid.
00066  */
00067 int uECC_verify(const uint8_t *public_key, const uint8_t *hash, unsigned hash_size,
00068                 const uint8_t *signature, const uECC_Curve_t *curve);
00069
00070 #ifdef __cplusplus
00071 }
```

```
00072 #endif
00073
00074 #endif /* UECC_H */
```

5.33 components/uECC/uECC_esp32.cpp File Reference

```
#include "uECC.h"
#include "CW_Utils.h"
#include "mbedtls/ecp.h"
#include "mbedtls/ecdsa.h"
#include "esp_log.h"
#include <algorithm>
```

Include dependency graph for uECC_esp32.cpp:



Classes

- struct uECC_Curve_t

Macros

- #define COORD_SIZE_BYTES (32U) /* bytes per coordinate (256-bit curve) */
- #define ECC_XY_KEY_SIZE (COORD_SIZE_BYTES * 2U) /* X[32] || Y[32] without 0x04 prefix */
- #define UNCOMPRESSED_PUB_SIZE (65U) /* 0x04 || X[32] || Y[32] */
- #define UNCOMPRESSED_PREFIX (0x04U)
- #define POINT_PREFIX_OFFSET (0U) /* byte offset of the 0x04 prefix */
- #define COORD_X_OFFSET (1U) /* byte offset of X in 65-byte key */
- #define RAW_SIG_R_OFFSET (0U) /* byte offset of r in 64-byte raw sig */
- #define RAW_SIG_S_OFFSET (32U) /* byte offset of s in 64-byte raw sig */
- #define UECC_SUCCESS (1)
- #define UECC_FAILURE (0)
- #define MBEDTLS_OK (0)
- #define RNG_ERROR (-1) /* returned by RNG callback on invalid arguments */

Functions

- static int esp32_mbedtls_rng (void *ctx, unsigned char *output, size_t len)
- const uECC_Curve_t * uECC_secp256r1 (void)

  Return the static secp256r1 curve descriptor.
- const uECC_Curve_t * uECC_secp256k1 (void)

  Return the static secp256k1 curve descriptor.
- void uECC_set_rng (uECC_RNG_Function rng_function)

  No-op: ESP32 hardware RNG is used internally; no external callback needed.

- int uECC_make_key (uint8_t *public_key, uint8_t *private_key, const uECC_Curve_t *curve)
  Generate an ECC key pair using mbedTLS and the ESP32 hardware RNG.
- int uECC_shared_secret (const uint8_t *public_key, const uint8_t *private_key, uint8_t *secret, const uECC_Curve_t *curve)
  *Compute ECDH shared secret (X-coordinate of privKey * pubKey).*
- int uECC_verify (const uint8_t *public_key, const uint8_t *hash, unsigned hash_size, const uint8_t *signature, const uECC_Curve_t *curve)
  Verify an ECDSA signature (raw 64-byte r||s) against a hash.

Variables

- static const char *const UECC_LOG_TAG = "uECC_esp32"
- static const uECC_Curve_t s_secp256r1 = { MBEDTLS_ECP_DP_SECP256R1 }
- static const uECC_Curve_t s_secp256k1 = { MBEDTLS_ECP_DP_SECP256K1 }

5.33.1 Macro Definition Documentation

5.33.1.1 COORD_SIZE_BYTES

```
#define COORD_SIZE_BYTES (32U) /* bytes per coordinate (256-bit curve) */
```
Definition at line 17 of file uECC_esp32.cpp.
Referenced by uECC_make_key(), uECC_shared_secret(), and uECC_verify().

5.33.1.2 COORD_X_OFFSET

```
#define COORD_X_OFFSET (1U) /* byte offset of X in 65-byte key */
```
Definition at line 23 of file uECC_esp32.cpp.
Referenced by uECC_make_key(), uECC_shared_secret(), and uECC_verify().

5.33.1.3 ECC_XY_KEY_SIZE

```
#define ECC_XY_KEY_SIZE (COORD_SIZE_BYTES * 2U) /* X[32] || Y[32] without 0x04 prefix */
```
Definition at line 19 of file uECC_esp32.cpp.
Referenced by uECC_make_key(), uECC_shared_secret(), and uECC_verify().

5.33.1.4 MBEDTLS_OK

```
#define MBEDTLS_OK (0)
```
Definition at line 28 of file uECC_esp32.cpp.

5.33.1.5 POINT_PREFIX_OFFSET

```
#define POINT_PREFIX_OFFSET (0U) /* byte offset of the 0x04 prefix */
```
Definition at line 22 of file uECC_esp32.cpp.
Referenced by uECC_shared_secret(), and uECC_verify().

5.33.1.6 RAW_SIG_R_OFFSET

```
#define RAW_SIG_R_OFFSET (0U) /* byte offset of r in 64-byte raw sig */
```
Definition at line 24 of file uECC_esp32.cpp.
Referenced by uECC_verify().

5.33.1.7 RAW_SIG_S_OFFSET

```
#define RAW_SIG_S_OFFSET (32U) /* byte offset of s in 64-byte raw sig */
```
Definition at line 25 of file uECC_esp32.cpp.
Referenced by uECC_verify().

5.33.1.8 RNG_ERROR

```
#define RNG_ERROR (-1) /* returned by RNG callback on invalid arguments */
```
Definition at line 29 of file uECC_esp32.cpp.
Referenced by esp32_mbedtls_rng().

5.33.1.9 UECC_FAILURE

```
#define UECC_FAILURE (0)
```
Definition at line 27 of file uECC_esp32.cpp.
Referenced by uECC_make_key(), uECC_shared_secret(), and uECC_verify().

5.33.1.10 UECC_SUCCESS

```
#define UECC_SUCCESS (1)
```
Definition at line 26 of file uECC_esp32.cpp.

5.33.1.11 UNCOMPRESSED_PREFIX

```
#define UNCOMPRESSED_PREFIX (0x04U)
```
Definition at line 21 of file uECC_esp32.cpp.
Referenced by uECC_shared_secret(), and uECC_verify().

5.33.1.12 UNCOMPRESSED_PUB_SIZE

```
#define UNCOMPRESSED_PUB_SIZE (65U) /* 0x04 || X[32] || Y[32] */
```
Definition at line 20 of file uECC_esp32.cpp.
Referenced by uECC_make_key(), uECC_shared_secret(), and uECC_verify().

5.33.2 Function Documentation

5.33.2.1 esp32_mbedtls_rng()

```
int esp32_mbedtls_rng (
            void * ctx,
            unsigned char * output,
            size_t len)  [static]
```
Definition at line 50 of file uECC_esp32.cpp.
References MBEDTLS_OK, and RNG_ERROR.
Referenced by uECC_make_key(), and uECC_shared_secret().

5.33.2.2 uECC_make_key()

```
int uECC_make_key (
            uint8_t * public_key,
            uint8_t * private_key,
            const uECC_Curve_t * curve)
```
Generate an ECC key pair using mbedTLS and the ESP32 hardware RNG.
Definition at line 92 of file uECC_esp32.cpp.
References COORD_SIZE_BYTES, COORD_X_OFFSET, ECC_XY_KEY_SIZE, esp32_mbedtls_rng(), uECC_Curve_t::grp_id, MBEDTLS_OK, UECC_FAILURE, UECC_SUCCESS, and UNCOMPRESSED_PUB_SIZE.
Referenced by ESP32CryptoProvider::makeKey().

5.33.2.3 uECC_secp256k1()

```
const uECC_Curve_t * uECC_secp256k1 (
            void )
```
Return the static secp256k1 curve descriptor.
Definition at line 75 of file uECC_esp32.cpp.
References s_secp256k1.

Referenced by toCurve().

5.33.2.4 uECC_secp256r1()

```
const uECC_Curve_t * uECC_secp256r1 (
            void )
```

Return the static secp256r1 curve descriptor.
Definition at line 70 of file uECC_esp32.cpp.
References s_secp256r1.
Referenced by toCurve().

5.33.2.5 uECC_set_rng()

```
void uECC_set_rng (
            uECC_RNG_Function rng_function)
```

No-op: ESP32 hardware RNG is used internally; no external callback needed.
Definition at line 80 of file uECC_esp32.cpp.
References UECC_LOG_TAG.

5.33.2.6 uECC_shared_secret()

```
int uECC_shared_secret (
            const uint8_t * public_key,
            const uint8_t * private_key,
            uint8_t * secret,
            const uECC_Curve_t * curve)
```

Compute ECDH shared secret (X-coordinate of privKey * pubKey).
Definition at line 144 of file uECC_esp32.cpp.
References COORD_SIZE_BYTES, COORD_X_OFFSET, ECC_XY_KEY_SIZE, esp32_mbedtls_rng(), uECC_Curve_t::grp_id, MBEDTLS_OK, POINT_PREFIX_OFFSET, UECC_FAILURE, UECC_SUCCESS, UNCOMPRESSED_PREFIX, and UNCOMPRESSED_PUB_SIZE.
Referenced by ESP32CryptoProvider::ecdh().

5.33.2.7 uECC_verify()

```
int uECC_verify (
            const uint8_t * public_key,
            const uint8_t * hash,
            unsigned hash_size,
            const uint8_t * signature,
            const uECC_Curve_t * curve)
```

Verify an ECDSA signature (raw 64-byte r||s) against a hash.
Definition at line 210 of file uECC_esp32.cpp.
References COORD_SIZE_BYTES, COORD_X_OFFSET, ECC_XY_KEY_SIZE, uECC_Curve_t::grp_id, MBEDTLS_OK, POINT_PREFIX_OFFSET, RAW_SIG_R_OFFSET, RAW_SIG_S_OFFSET, UECC_FAILURE, UECC_SUCCESS, UNCOMPRESSED_PREFIX, and UNCOMPRESSED_PUB_SIZE.
Referenced by ESP32CryptoProvider::ecdsaVerify().

5.33.3 Variable Documentation

5.33.3.1 s_secp256k1

```
const uECC_Curve_t s_secp256k1 = { MBEDTLS_ECP_DP_SECP256K1 }  [static]
```

Definition at line 44 of file uECC_esp32.cpp.
Referenced by uECC_secp256k1().

5.33.3.2 s_secp256r1

```
const uECC_Curve_t s_secp256r1 = { MBEDTLS_ECP_DP_SECP256R1 }  [static]
```

Definition at line 43 of file uECC_esp32.cpp.
Referenced by uECC_secp256r1().

5.33.3.3 UECC_LOG_TAG

```
const char* const UECC_LOG_TAG = "uECC_esp32"  [static]
```

Definition at line 18 of file uECC_esp32.cpp.
Referenced by uECC_set_rng().

5.34 uECC_esp32.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #include "uECC.h"
00007 #include "CW_Utils.h"
00008 #include "mbedtls/ecp.h"
00009 #include "mbedtls/ecdsa.h"
00010 #include "esp_log.h"
00011 #include <algorithm>
00012
00013 /*****************************************************************
00014  * 1. Module constants
00015  *****************************************************************/
00016
00017 #define COORD_SIZE_BYTES       (32U)                      /* bytes per coordinate (256-bit curve) */
00018 static const char* const UECC_LOG_TAG = "uECC_esp32";
00019 #define ECC_XY_KEY_SIZE        (COORD_SIZE_BYTES * 2U)    /* X[32] || Y[32] without 0x04 prefix  */
00020 #define UNCOMPRESSED_PUB_SIZE  (65U)                      /* 0x04 || X[32] || Y[32]              */
00021 #define UNCOMPRESSED_PREFIX    (0x04U)
00022 #define POINT_PREFIX_OFFSET    (0U)   /* byte offset of the 0x04 prefix      */
00023 #define COORD_X_OFFSET         (1U)   /* byte offset of X in 65-byte key     */
00024 #define RAW_SIG_R_OFFSET       (0U)   /* byte offset of r in 64-byte raw sig */
00025 #define RAW_SIG_S_OFFSET       (32U)  /* byte offset of s in 64-byte raw sig */
00026 #define UECC_SUCCESS           (1)
00027 #define UECC_FAILURE           (0)
00028 #define MBEDTLS_OK             (0)
00029 #define RNG_ERROR              (-1)  /* returned by RNG callback on invalid arguments */
00030
00031 /*****************************************************************
00032  * 2. uECC_Curve_t definition (opaque in uECC.h)
00033  *****************************************************************/
00034
00035 struct uECC_Curve_t {
00036     mbedtls_ecp_group_id grp_id;
00037 };
00038
00039 /*****************************************************************
00040  * 3. Static curve instances
00041  *****************************************************************/
00042
00043 static const uECC_Curve_t s_secp256r1 = { MBEDTLS_ECP_DP_SECP256R1 };
00044 static const uECC_Curve_t s_secp256k1 = { MBEDTLS_ECP_DP_SECP256K1 };
00045
00046 /*****************************************************************
00047  * 4. Internal RNG adapter for mbedTLS
00048  *****************************************************************/
00049
00050 static int esp32_mbedtls_rng(void *ctx, unsigned char *output, size_t len) {
00051     int result = RNG_ERROR;
00052     (void)ctx;
00053
00054     if ((output != NULL) && (len > 0U)) {
00055         bool rng_result = CW_Utils::fill_secure_random(
00056                               reinterpret_cast<uint8_t *>(output), len);
00057         if (rng_result) {
00058             result = MBEDTLS_OK;
00059         }
00060     }
00061
00062     return result;
00063 }
00064
00065 /*****************************************************************
00066  * 5. Public API
00067  *****************************************************************/
00068
00070 const uECC_Curve_t* uECC_secp256r1(void) {
00071     return &s_secp256r1;
00072 }
00073
00075 const uECC_Curve_t* uECC_secp256k1(void) {
```

```
00076     return &s_secp256k1;
00077 }
00078
00080 void uECC_set_rng(uECC_RNG_Function rng_function) {
00081     if (rng_function != NULL) {
00082         /* The caller's RNG callback is intentionally ignored: ESP32 routes all
00083          * entropy through mbedTLS and the hardware TRNG directly (SEC-018).
00084          * Log once so the drop is visible in debug output rather than silent. */
00085         ESP_LOGW(UECC_LOG_TAG,
00086                  "uECC_set_rng: callback ignored -- ESP32 uses hardware RNG internally");
00087     }
00088     (void)rng_function;
00089 }
00090
00092 int uECC_make_key(uint8_t *public_key, uint8_t *private_key,
00093                   const uECC_Curve_t *curve) {
00094     int result = UECC_FAILURE;
00095
00096     if ((public_key != NULL) && (private_key != NULL) && (curve != NULL)) {
00097         mbedtls_ecp_group grp = {};
00098         mbedtls_mpi       d   = {};
00099         mbedtls_ecp_point Q   = {};
00100
00101         mbedtls_ecp_group_init(&grp);
00102         mbedtls_mpi_init(&d);
00103         mbedtls_ecp_point_init(&Q);
00104
00105         int ret = mbedtls_ecp_group_load(&grp, curve->grp_id);
00106
00107         if (ret == MBEDTLS_OK) {
00108             ret = mbedtls_ecp_gen_keypair(&grp, &d, &Q,
00109                                           esp32_mbedtls_rng, NULL);
00110         }
00111
00112         if (ret == MBEDTLS_OK) {
00113             ret = mbedtls_mpi_write_binary(&d, private_key,
00114                                            static_cast<size_t>(COORD_SIZE_BYTES));
00115         }
00116
00117         if (ret == MBEDTLS_OK) {
00118             uint8_t pub65[UNCOMPRESSED_PUB_SIZE] = { 0U };
00119             size_t  olen = 0U;
00120             ret = mbedtls_ecp_point_write_binary(&grp, &Q,
00121                                                  MBEDTLS_ECP_PF_UNCOMPRESSED,
00122                                                  &olen,
00123                                                  pub65, sizeof(pub65));
00124             if (ret == MBEDTLS_OK) {
00125                 (void)std::copy_n(pub65 + COORD_X_OFFSET,
00126                                   static_cast<size_t>(ECC_XY_KEY_SIZE),
00127                                   public_key);
00128             }
00129         }
00130
00131         if (ret == MBEDTLS_OK) {
00132             result = UECC_SUCCESS;
00133         }
00134
00135         mbedtls_ecp_point_free(&Q);
00136         mbedtls_mpi_free(&d);
00137         mbedtls_ecp_group_free(&grp);
00138     }
00139
00140     return result;
00141 }
00142
00144 int uECC_shared_secret(const uint8_t *public_key, const uint8_t *private_key,
00145                        uint8_t *secret, const uECC_Curve_t *curve) {
00146     int result = UECC_FAILURE;
00147
00148     if ((public_key != NULL) && (private_key != NULL) &&
00149         (secret     != NULL) && (curve       != NULL)) {
00150         mbedtls_ecp_group grp      = {};
00151         mbedtls_ecp_point remote_Q = {};
00152         mbedtls_mpi       local_d  = {};
00153         mbedtls_ecp_point shared_R = {};
00154
00155         mbedtls_ecp_group_init(&grp);
00156         mbedtls_ecp_point_init(&remote_Q);
00157         mbedtls_mpi_init(&local_d);
00158         mbedtls_ecp_point_init(&shared_R);
00159
00160         int ret = mbedtls_ecp_group_load(&grp, curve->grp_id);
00161
00162         if (ret == MBEDTLS_OK) {
00163             uint8_t pub65[UNCOMPRESSED_PUB_SIZE] = { 0U };
00164             pub65[POINT_PREFIX_OFFSET] = UNCOMPRESSED_PREFIX;
00165             (void)std::copy_n(public_key,
```

```
00166                               static_cast<size_t>(ECC_XY_KEY_SIZE),
00167                               pub65 + COORD_X_OFFSET);
00168             ret = mbedtls_ecp_point_read_binary(&grp, &remote_Q,
00169                                                 pub65, sizeof(pub65));
00170         }
00171
00172         if (ret == MBEDTLS_OK) {
00173             ret = mbedtls_mpi_read_binary(&local_d, private_key,
00174                                           static_cast<size_t>(COORD_SIZE_BYTES));
00175         }
00176
00177         if (ret == MBEDTLS_OK) {
00178             ret = mbedtls_ecp_mul(&grp, &shared_R, &local_d, &remote_Q,
00179                                   esp32_mbedtls_rng, NULL);
00180         }
00181
00182         if (ret == MBEDTLS_OK) {
00183             uint8_t shared65[UNCOMPRESSED_PUB_SIZE] = { 0U };
00184             size_t  olen = 0U;
00185             ret = mbedtls_ecp_point_write_binary(&grp, &shared_R,
00186                                                  MBEDTLS_ECP_PF_UNCOMPRESSED,
00187                                                  &olen,
00188                                                  shared65, sizeof(shared65));
00189             if (ret == MBEDTLS_OK) {
00190                 (void)std::copy_n(shared65 + COORD_X_OFFSET,
00191                                   static_cast<size_t>(COORD_SIZE_BYTES),
00192                                   secret);
00193             }
00194         }
00195
00196         if (ret == MBEDTLS_OK) {
00197             result = UECC_SUCCESS;
00198         }
00199
00200         mbedtls_ecp_point_free(&shared_R);
00201         mbedtls_mpi_free(&local_d);
00202         mbedtls_ecp_point_free(&remote_Q);
00203         mbedtls_ecp_group_free(&grp);
00204     }
00205
00206     return result;
00207 }
00208
00210 int uECC_verify(const uint8_t *public_key, const uint8_t *hash, unsigned hash_size,
00211                 const uint8_t *signature, const uECC_Curve_t *curve) {
00212     int result = UECC_FAILURE;
00213
00214     if ((public_key != NULL) && (hash != NULL) &&
00215         (signature  != NULL) && (curve != NULL)) {
00216         mbedtls_ecp_group grp = {};
00217         mbedtls_ecp_point Q   = {};
00218         mbedtls_mpi       r   = {};
00219         mbedtls_mpi       s   = {};
00220
00221         mbedtls_ecp_group_init(&grp);
00222         mbedtls_ecp_point_init(&Q);
00223         mbedtls_mpi_init(&r);
00224         mbedtls_mpi_init(&s);
00225
00226         int ret = mbedtls_ecp_group_load(&grp, curve->grp_id);
00227
00228         if (ret == MBEDTLS_OK) {
00229             uint8_t pub65[UNCOMPRESSED_PUB_SIZE] = { 0U };
00230             pub65[POINT_PREFIX_OFFSET] = UNCOMPRESSED_PREFIX;
00231             (void)std::copy_n(public_key,
00232                               static_cast<size_t>(ECC_XY_KEY_SIZE),
00233                               pub65 + COORD_X_OFFSET);
00234             ret = mbedtls_ecp_point_read_binary(&grp, &Q,
00235                                                 pub65, sizeof(pub65));
00236         }
00237
00238         if (ret == MBEDTLS_OK) {
00239             ret = mbedtls_mpi_read_binary(&r,
00240                                           signature + RAW_SIG_R_OFFSET,
00241                                           static_cast<size_t>(COORD_SIZE_BYTES));
00242         }
00243
00244         if (ret == MBEDTLS_OK) {
00245             ret = mbedtls_mpi_read_binary(&s,
00246                                           signature + RAW_SIG_S_OFFSET,
00247                                           static_cast<size_t>(COORD_SIZE_BYTES));
00248         }
00249
00250         if (ret == MBEDTLS_OK) {
00251             ret = mbedtls_ecdsa_verify(&grp, hash,
00252                                        static_cast<size_t>(hash_size),
00253                                        &Q, &r, &s);
```

```
00254         }
00255
00256         if (ret == MBEDTLS_OK) {
00257             result = UECC_SUCCESS;
00258         }
00259
00260         mbedtls_mpi_free(&s);
00261         mbedtls_mpi_free(&r);
00262         mbedtls_ecp_point_free(&Q);
00263         mbedtls_ecp_group_free(&grp);
00264     }
00265
00266     return result;
00267 }
```

5.35 examples/BasicUsage/README.md File Reference

5.36 examples/Connect/README.md File Reference

5.37 examples/README.md File Reference

5.38 examples/Sign/README.md File Reference

5.39 examples/UsdcSigning/README.md File Reference

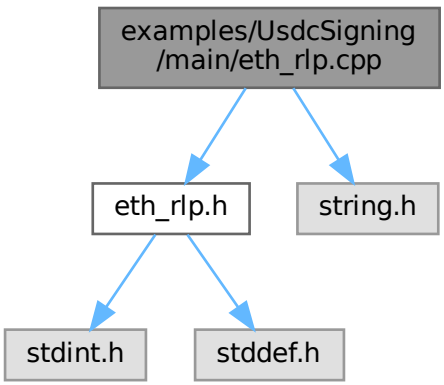
5.40 examples/VerifyPin/README.md File Reference

5.41 README.md File Reference

5.42 examples/UsdcSigning/main/eth_rlp.cpp File Reference

```
#include "eth_rlp.h"
#include <string.h>
```

Include dependency graph for eth_rlp.cpp:



Macros

- #define ITEMS_BUF_MAX 320U

Functions

- static uint8_t be_minimal (uint64_t value, uint8_t out[8])
- static size_t rlp_bytes (const uint8_t *data, size_t data_len, uint8_t *out)
- static size_t rlp_uint64 (uint64_t value, uint8_t *out)
- static size_t rlp_int256 (const uint8_t data[32], uint8_t *out)
- static size_t rlp_list_header (size_t content_len, uint8_t *out)
- static size_t encode_common_fields (const eth_tx_t *tx, uint8_t *items)
- size_t eth_rlp_encode_unsigned (const eth_tx_t *tx, uint8_t *out, size_t out_max)
- size_t eth_rlp_encode_signed (const eth_tx_t *tx, uint8_t v, const uint8_t r[32], const uint8_t s[32], uint8_t *out, size_t out_max)

5.42.1 Macro Definition Documentation

5.42.1.1 ITEMS_BUF_MAX

```
#define ITEMS_BUF_MAX 320U
```

Definition at line 10 of file eth_rlp.cpp.
Referenced by eth_rlp_encode_signed(), and eth_rlp_encode_unsigned().

5.42.2 Function Documentation

5.42.2.1 be_minimal()

```
uint8_t be_minimal (
            uint64_t value,
            uint8_t out[8])  [static]
```

Definition at line 22 of file eth_rlp.cpp.
Referenced by rlp_bytes(), rlp_list_header(), and rlp_uint64().

5.42.2.2 encode_common_fields()

```
size_t encode_common_fields (
            const eth_tx_t * tx,
            uint8_t * items)  [static]
```

Definition at line 133 of file eth_rlp.cpp.
References eth_tx_t::calldata, eth_tx_t::calldata_len, eth_tx_t::chain_id, eth_tx_t::eth_value, eth_tx_t::gas_limit, eth_tx_t::max_fee, eth_tx_t::max_priority_fee, eth_tx_t::nonce, rlp_bytes(), rlp_uint64(), and eth_tx_t::to.
Referenced by eth_rlp_encode_signed(), and eth_rlp_encode_unsigned().

5.42.2.3 eth_rlp_encode_signed()

```
size_t eth_rlp_encode_signed (
            const eth_tx_t * tx,
            uint8_t v,
            const uint8_t r[32],
            const uint8_t s[32],
            uint8_t * out,
            size_t out_max)
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 169 of file eth_rlp.cpp.
References encode_common_fields(), ITEMS_BUF_MAX, rlp_int256(), rlp_list_header(), and rlp_uint64().
Referenced by signing_loop().

5.42.2.4 eth_rlp_encode_unsigned()

```
size_t eth_rlp_encode_unsigned (
            const eth_tx_t * tx,
            uint8_t * out,
            size_t out_max)
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 152 of file eth_rlp.cpp.
References encode_common_fields(), ITEMS_BUF_MAX, and rlp_list_header().
Referenced by signing_loop().

5.42.2.5 rlp_bytes()

```
size_t rlp_bytes (
            const uint8_t * data,
            size_t data_len,
            uint8_t * out)  [static]
```

Definition at line 51 of file eth_rlp.cpp.
References be_minimal().
Referenced by encode_common_fields(), rlp_int256(), and rlp_uint64().

5.42.2.6 rlp_int256()

```
size_t rlp_int256 (
            const uint8_t data[32],
            uint8_t * out)  [static]
```

Definition at line 98 of file eth_rlp.cpp.
References rlp_bytes().
Referenced by eth_rlp_encode_signed().

5.42.2.7 rlp_list_header()

```
size_t rlp_list_header (
            size_t content_len,
            uint8_t * out)  [static]
```

Definition at line 116 of file eth_rlp.cpp.
References be_minimal().
Referenced by eth_rlp_encode_signed(), and eth_rlp_encode_unsigned().

5.42.2.8 rlp_uint64()

```
size_t rlp_uint64 (
            uint64_t value,
            uint8_t * out)  [static]
```

Definition at line 82 of file eth_rlp.cpp.
References be_minimal(), and rlp_bytes().
Referenced by encode_common_fields(), and eth_rlp_encode_signed().

5.43 eth_rlp.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #include "eth_rlp.h"
00007 #include <string.h>
00008
00009 /* Maximum scratch buffer for the list content before prepending the header. */
```

```
00010 #define ITEMS_BUF_MAX  320U
00011
00012 /**************************************************************
00013  * Internal helpers
00014  **************************************************************/
00015
00016 /*
00017  * Write 'value' as a minimal big-endian byte sequence into out[0..].
00018  * Returns the number of bytes written (1-8).  Value 0 produces one 0x00 byte;
00019  * the caller is responsible for treating the empty-integer case (value==0
00020  * → RLP 0x80) before calling this function.
00021  */
00022 static uint8_t be_minimal(uint64_t value, uint8_t out[8])
00023 {
00024     uint8_t tmp[8];
00025     uint8_t first = 0U;
00026     uint8_t i;
00027
00028     tmp[0] = (uint8_t)((value >> 56U) & 0xFFU);
00029     tmp[1] = (uint8_t)((value >> 48U) & 0xFFU);
00030     tmp[2] = (uint8_t)((value >> 40U) & 0xFFU);
00031     tmp[3] = (uint8_t)((value >> 32U) & 0xFFU);
00032     tmp[4] = (uint8_t)((value >> 24U) & 0xFFU);
00033     tmp[5] = (uint8_t)((value >> 16U) & 0xFFU);
00034     tmp[6] = (uint8_t)((value >>  8U) & 0xFFU);
00035     tmp[7] = (uint8_t)(value & 0xFFU);
00036
00037     while ((first < 7U) && (tmp[first] == 0U)) {
00038         first++;
00039     }
00040     uint8_t count = (uint8_t)(8U - first);
00041     for (i = 0U; i < count; i++) {
00042         out[i] = tmp[(size_t)first + i];
00043     }
00044     return count;
00045 }
00046
00047 /*
00048  * RLP-encode a byte string of length data_len at 'data' into out.
00049  * Returns bytes written.
00050  */
00051 static size_t rlp_bytes(const uint8_t *data, size_t data_len, uint8_t *out)
00052 {
00053     size_t written = 0U;
00054
00055     if (data_len == 0U) {
00056         out[0] = 0x80U;
00057         written = 1U;
00058     } else if ((data_len == 1U) && (data[0] < 0x80U)) {
00059         out[0] = data[0];
00060         written = 1U;
00061     } else if (data_len <= 55U) {
00062         out[0] = (uint8_t)(0x80U + data_len);
00063         (void)memcpy(out + 1U, data, data_len);
00064         written = 1U + data_len;
00065     } else {
00066         /* Long string: 0xb7 + len(length) bytes, then length, then data. */
00067         uint8_t len_bytes[8];
00068         uint8_t num_lb = be_minimal((uint64_t)data_len, len_bytes);
00069         out[0] = (uint8_t)(0xB7U + num_lb);
00070         (void)memcpy(out + 1U, len_bytes, num_lb);
00071         (void)memcpy(out + 1U + num_lb, data, data_len);
00072         written = 1U + num_lb + data_len;
00073     }
00074     return written;
00075 }
00076
00077 /*
00078  * RLP-encode a non-negative integer (uint64_t) into out.
00079  * 0 is encoded as the empty byte string (0x80).
00080  * Returns bytes written.
00081  */
00082 static size_t rlp_uint64(uint64_t value, uint8_t *out)
00083 {
00084     if (value == 0U) {
00085         out[0] = 0x80U;
00086         return 1U;
00087     }
00088     uint8_t be[8];
00089     uint8_t be_len = be_minimal(value, be);
00090     return rlp_bytes(be, be_len, out);
00091 }
00092
00093 /*
00094  * RLP-encode a 32-byte big-endian integer (e.g. r or s from ECDSA) into out.
00095  * Leading zero bytes are stripped so the encoding is minimal.
00096  * Returns bytes written.
```

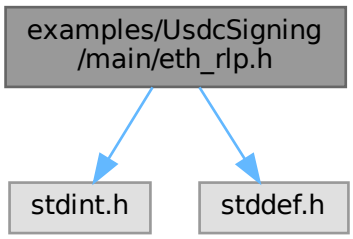
```
00097  */
00098 static size_t rlp_int256(const uint8_t data[32], uint8_t *out)
00099 {
00100     size_t first = 0U;
00101     while ((first < 31U) && (data[first] == 0U)) {
00102         first++;
00103     }
00104     if ((first == 31U) && (data[31] == 0U)) {
00105         /* Value is zero */
00106         out[0] = 0x80U;
00107         return 1U;
00108     }
00109     return rlp_bytes(data + first, 32U - first, out);
00110 }
00111
00112 /*
00113  * Write the RLP list header for a list whose content is content_len bytes.
00114  * Returns bytes written (does NOT write the content).
00115  */
00116 static size_t rlp_list_header(size_t content_len, uint8_t *out)
00117 {
00118     if (content_len <= 55U) {
00119         out[0] = (uint8_t)(0xC0U + content_len);
00120         return 1U;
00121     }
00122     uint8_t len_bytes[8];
00123     uint8_t num_lb = be_minimal((uint64_t)content_len, len_bytes);
00124     out[0] = (uint8_t)(0xF7U + num_lb);
00125     (void)memcpy(out + 1U, len_bytes, num_lb);
00126     return 1U + num_lb;
00127 }
00128
00129 /*****************************************************************
00130  * Shared field encoder
00131  *****************************************************************/
00132
00133 static size_t encode_common_fields(const eth_tx_t *tx, uint8_t *items)
00134 {
00135     size_t n = 0U;
00136     n += rlp_uint64(tx->chain_id,          items + n);
00137     n += rlp_uint64(tx->nonce,             items + n);
00138     n += rlp_uint64(tx->max_priority_fee,  items + n);
00139     n += rlp_uint64(tx->max_fee,           items + n);
00140     n += rlp_uint64(tx->gas_limit,         items + n);
00141     n += rlp_bytes(tx->to, 20U,            items + n);
00142     n += rlp_uint64(tx->eth_value,         items + n);
00143     n += rlp_bytes(tx->calldata, tx->calldata_len, items + n);
00144     items[n++] = 0xC0U;   /* empty access list */
00145     return n;
00146 }
00147
00148 /*****************************************************************
00149  * Public API
00150  *****************************************************************/
00151
00152 size_t eth_rlp_encode_unsigned(const eth_tx_t *tx, uint8_t *out, size_t out_max)
00153 {
00154     uint8_t items[ITEMS_BUF_MAX];
00155     size_t  items_len = encode_common_fields(tx, items);
00156
00157     uint8_t hdr[10];
00158     size_t  hdr_len = rlp_list_header(items_len, hdr);
00159
00160     size_t total = 1U + hdr_len + items_len;   /* 0x02 prefix + list */
00161     if (total > out_max) { return 0U; }
00162
00163     out[0] = 0x02U;                            /* EIP-1559 type prefix */
00164     (void)memcpy(out + 1U,          hdr,   hdr_len);
00165     (void)memcpy(out + 1U + hdr_len, items, items_len);
00166     return total;
00167 }
00168
00169 size_t eth_rlp_encode_signed(const eth_tx_t *tx, uint8_t v,
00170                              const uint8_t r[32], const uint8_t s[32],
00171                              uint8_t *out, size_t out_max)
00172 {
00173     uint8_t items[ITEMS_BUF_MAX];
00174     size_t  items_len = encode_common_fields(tx, items);
00175
00176     items_len += rlp_uint64((uint64_t)v,  items + items_len);
00177     items_len += rlp_int256(r,            items + items_len);
00178     items_len += rlp_int256(s,            items + items_len);
00179
00180     uint8_t hdr[10];
00181     size_t  hdr_len = rlp_list_header(items_len, hdr);
00182
00183     size_t total = 1U + hdr_len + items_len;
```

```
00184     if (total > out_max) { return 0U; }
00185
00186     out[0] = 0x02U;
00187     (void)memcpy(out + 1U,           hdr,   hdr_len);
00188     (void)memcpy(out + 1U + hdr_len, items, items_len);
00189     return total;
00190 }
```

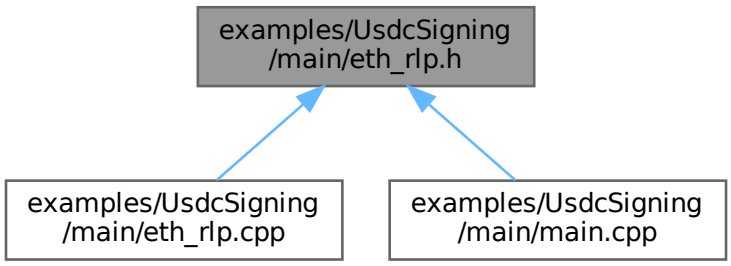
5.44 examples/UsdcSigning/main/eth_rlp.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for eth_rlp.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct eth_tx_t

Functions

- size_t eth_rlp_encode_unsigned (const eth_tx_t *tx, uint8_t *out, size_t out_max)
- size_t eth_rlp_encode_signed (const eth_tx_t *tx, uint8_t v, const uint8_t r[32], const uint8_↩ t s[32], uint8_t *out, size_t out_max)

5.44.1 Function Documentation

5.44.1.1 eth_rlp_encode_signed()

```
size_t eth_rlp_encode_signed (
            const eth_tx_t * tx,
            uint8_t v,
            const uint8_t r[32],
            const uint8_t s[32],
            uint8_t * out,
            size_t out_max)
```

Definition at line 169 of file eth_rlp.cpp.
References encode_common_fields(), ITEMS_BUF_MAX, rlp_int256(), rlp_list_header(), and rlp_uint64().
Referenced by signing_loop().

5.44.1.2 eth_rlp_encode_unsigned()

```
size_t eth_rlp_encode_unsigned (
            const eth_tx_t * tx,
            uint8_t * out,
            size_t out_max)
```

Definition at line 152 of file eth_rlp.cpp.
References encode_common_fields(), ITEMS_BUF_MAX, and rlp_list_header().
Referenced by signing_loop().

5.45 eth_rlp.h

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #pragma once
00007
00008 #include <stdint.h>
00009 #include <stddef.h>
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014
00015 /* EIP-1559 (type 2) transaction parameters. */
00016 typedef struct {
00017     uint64_t       chain_id;
00018     uint64_t       nonce;
00019     uint64_t       max_priority_fee;  /* wei */
00020     uint64_t       max_fee;           /* wei */
00021     uint64_t       gas_limit;
00022     uint8_t        to[20];            /* recipient Ethereum address */
00023     uint64_t       eth_value;         /* wei (0 for pure ERC-20 transfer) */
00024     const uint8_t *calldata;
00025     size_t         calldata_len;
00026 } eth_tx_t;
00027
00028 /*
00029  * Encode an unsigned EIP-1559 transaction: 0x02 || RLP([chainId, nonce, ...]).
00030  * Returns total bytes written, or 0 on overflow.
00031  */
00032 size_t eth_rlp_encode_unsigned(const eth_tx_t *tx, uint8_t *out, size_t out_max);
00033
00034 /*
00035  * Encode a signed EIP-1559 transaction: 0x02 || RLP([..., v, r, s]).
00036  * v must be 0 or 1.  r and s are 32-byte big-endian values.
00037  * Returns total bytes written, or 0 on overflow.
00038  */
00039 size_t eth_rlp_encode_signed(const eth_tx_t *tx, uint8_t v,
00040                              const uint8_t r[32], const uint8_t s[32],
00041                              uint8_t *out, size_t out_max);
00042
00043 #ifdef __cplusplus
00044 }
00045 #endif
```

5.46 examples/UsdcSigning/main/eth_rpc.cpp File Reference

```
#include "eth_rpc.h"
#include <string.h>
#include <strings.h>
#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_netif.h"
#include "esp_log.h"
#include "esp_http_client.h"
#include "esp_crt_bundle.h"
```

Include dependency graph for eth_rpc.cpp:

Macros

- #define WIFI_CONNECTED_BIT BIT0
- #define WIFI_FAIL_BIT BIT1
- #define WIFI_MAX_RETRY 5
- #define WIFI_TIMEOUT_MS 30000
- #define RESP_BUF_SIZE 1024U
- #define HEX_PER_BYTE 2U

Functions

- static void wifi_event_handler (void *arg, esp_event_base_t event_base, int32_t event_id, void *event_data)
- static bool do_post (const char *body, char *resp_buf, size_t resp_buf_size)
- static char hex_nibble (uint8_t n)
- static void bytes_to_hex (const uint8_t *data, size_t len, char *out)
- void eth_rpc_init (const char *rpc_url, const char *from_addr)
- void eth_rpc_set_auth (const char *project_id, const char *api_secret)
- bool eth_rpc_wifi_connect (const char *ssid, const char *password)
- bool eth_rpc_get_nonce (uint64_t *nonce_out)
- uint8_t eth_rpc_ecrecover_parity (const uint8_t hash[32], const uint8_t r[32], const uint8_t s[32])
- bool eth_rpc_send_raw_tx (const uint8_t *tx, size_t tx_len, char *tx_hash_out, size_t tx_hash_↩ max)

Variables

- static const char *const TAG = "eth_rpc"
- static const char * s_rpc_url = NULL
- static const char * s_from_addr = NULL
- static const char * s_project_id = NULL
- static const char * s_api_secret = NULL
- static EventGroupHandle_t s_wifi_event_group
- static int s_retry_num = 0

5.46.1 Macro Definition Documentation

5.46.1.1 HEX_PER_BYTE

```
#define HEX_PER_BYTE 2U
```

Definition at line 34 of file eth_rpc.cpp.
Referenced by bytes_to_hex(), eth_rpc_ecrecover_parity(), and eth_rpc_send_raw_tx().

5.46.1.2 RESP_BUF_SIZE

```
#define RESP_BUF_SIZE 1024U
```

Definition at line 31 of file eth_rpc.cpp.
Referenced by eth_rpc_ecrecover_parity(), eth_rpc_get_nonce(), and eth_rpc_send_raw_tx().

5.46.1.3 WIFI_CONNECTED_BIT

```
#define WIFI_CONNECTED_BIT BIT0
```

Definition at line 25 of file eth_rpc.cpp.

5.46.1.4 WIFI_FAIL_BIT

```
#define WIFI_FAIL_BIT BIT1
```

Definition at line 26 of file eth_rpc.cpp.

5.46.1.5 WIFI_MAX_RETRY

```
#define WIFI_MAX_RETRY 5
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 27 of file eth_rpc.cpp.
Referenced by wifi_event_handler(), and wifi_event_handler().

5.46.1.6 WIFI_TIMEOUT_MS

```
#define WIFI_TIMEOUT_MS 30000
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 28 of file eth_rpc.cpp.
Referenced by eth_rpc_wifi_connect(), and wifi_start().

5.46.2 Function Documentation

5.46.2.1 bytes_to_hex()

```
void bytes_to_hex (
            const uint8_t * data,
            size_t len,
            char * out)  [static]
```

Definition at line 156 of file eth_rpc.cpp.
References hex_nibble(), and HEX_PER_BYTE.
Referenced by eth_rpc_ecrecover_parity(), and eth_rpc_send_raw_tx().

5.46.2.2 do_post()

```
bool do_post (
            const char * body,
            char * resp_buf,
            size_t resp_buf_size)  [static]
```

Definition at line 85 of file eth_rpc.cpp.
References s_api_secret, s_project_id, s_rpc_url, and TAG.
Referenced by eth_rpc_ecrecover_parity(), eth_rpc_get_nonce(), and eth_rpc_send_raw_tx().

5.46.2.3 eth_rpc_ecrecover_parity()

```
uint8_t eth_rpc_ecrecover_parity (
            const uint8_t hash[32],
            const uint8_t r[32],
            const uint8_t s[32])
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 257 of file eth_rpc.cpp.
References bytes_to_hex(), do_post(), hex_nibble(), HEX_PER_BYTE, RESP_BUF_SIZE, s_from_addr, and TAG.
Referenced by signing_loop().

5.46.2.4 eth_rpc_get_nonce()

```
bool eth_rpc_get_nonce (
            uint64_t * nonce_out)
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 231 of file eth_rpc.cpp.
References do_post(), RESP_BUF_SIZE, s_from_addr, and TAG.
Referenced by signing_loop().

5.46.2.5 eth_rpc_init()

```
void eth_rpc_init (
            const char * rpc_url,
            const char * from_addr)
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 169 of file eth_rpc.cpp.
References s_from_addr, and s_rpc_url.
Referenced by app_main().

5.46.2.6 eth_rpc_send_raw_tx()

```
bool eth_rpc_send_raw_tx (
            const uint8_t * tx,
            size_t tx_len,
            char * tx_hash_out,
            size_t tx_hash_max)
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 332 of file eth_rpc.cpp.
References bytes_to_hex(), do_post(), HEX_PER_BYTE, RESP_BUF_SIZE, and TAG.
Referenced by signing_loop().

5.46.2.7 eth_rpc_set_auth()

```
void eth_rpc_set_auth (
            const char * project_id,
            const char * api_secret)
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 175 of file eth_rpc.cpp.
References s_api_secret, and s_project_id.
Referenced by app_main().

5.46.2.8 eth_rpc_wifi_connect()

```
bool eth_rpc_wifi_connect (
            const char * ssid,
            const char * password)
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 181 of file eth_rpc.cpp.
References s_retry_num, s_wifi_event_group, TAG, WIFI_CONNECTED_BIT, wifi_event_handler(), WIFI_FAIL_BIT, and WIFI_TIMEOUT_MS.
Referenced by app_main().

5.46.2.9 hex_nibble()

```
char hex_nibble (
            uint8_t n)  [static]
```

Definition at line 151 of file eth_rpc.cpp.
Referenced by bytes_to_hex(), and eth_rpc_ecrecover_parity().

5.46.2.10 wifi_event_handler()

```
void wifi_event_handler (
            void * arg,
            esp_event_base_t event_base,
            int32_t event_id,
            void * event_data)  [static]
```

Definition at line 52 of file eth_rpc.cpp.
References s_retry_num, s_wifi_event_group, TAG, WIFI_CONNECTED_BIT, WIFI_FAIL_BIT, and WIFI_MAX_RETRY.

5.46.3 Variable Documentation

5.46.3.1 s_api_secret

```
const char* s_api_secret = NULL  [static]
```

Definition at line 43 of file eth_rpc.cpp.
Referenced by do_post(), and eth_rpc_set_auth().

5.46.3.2 s_from_addr

```
const char* s_from_addr = NULL  [static]
```

Definition at line 41 of file eth_rpc.cpp.
Referenced by eth_rpc_ecrecover_parity(), eth_rpc_get_nonce(), and eth_rpc_init().

5.46.3.3 s_project_id

```
const char* s_project_id = NULL  [static]
```
Definition at line 42 of file eth_rpc.cpp.
Referenced by do_post(), and eth_rpc_set_auth().

5.46.3.4 s_retry_num

```
int s_retry_num = 0  [static]
```
Definition at line 46 of file eth_rpc.cpp.

5.46.3.5 s_rpc_url

```
const char* s_rpc_url = NULL  [static]
```
Definition at line 40 of file eth_rpc.cpp.
Referenced by do_post(), and eth_rpc_init().

5.46.3.6 s_wifi_event_group

```
EventGroupHandle_t s_wifi_event_group  [static]
```
Definition at line 45 of file eth_rpc.cpp.

5.46.3.7 TAG

```
const char* const TAG = "eth_rpc"  [static]
```
Definition at line 23 of file eth_rpc.cpp.

5.47 eth_rpc.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #include "eth_rpc.h"
00007
00008 #include <string.h>
00009 #include <strings.h>    /* strncasecmp */
00010 #include <stdlib.h>     /* strtoull, malloc, free */
00011 #include <stdio.h>      /* snprintf */
00012 #include <inttypes.h>   /* PRIu64 */
00013
00014 #include "freertos/FreeRTOS.h"
00015 #include "freertos/event_groups.h"
00016 #include "esp_wifi.h"
00017 #include "esp_event.h"
00018 #include "esp_netif.h"
00019 #include "esp_log.h"
00020 #include "esp_http_client.h"
00021 #include "esp_crt_bundle.h"
00022
00023 static const char *const TAG = "eth_rpc";
00024
00025 #define WIFI_CONNECTED_BIT  BIT0
00026 #define WIFI_FAIL_BIT       BIT1
00027 #define WIFI_MAX_RETRY      5
00028 #define WIFI_TIMEOUT_MS     30000
00029
00030 /* JSON-RPC response buffer -- large enough for any expected response. */
00031 #define RESP_BUF_SIZE  1024U
00032
00033 /* Hex chars per byte */
00034 #define HEX_PER_BYTE  2U
00035
00036 /*********************************************************************
00037  * Module state
00038  ********************************************************************/
00039
00040 static const char *s_rpc_url      = NULL;
00041 static const char *s_from_addr    = NULL;
```

```
00042 static const char *s_project_id   = NULL;
00043 static const char *s_api_secret   = NULL;
00044
00045 static EventGroupHandle_t s_wifi_event_group;
00046 static int                s_retry_num = 0;
00047
00048 /****************************************************************
00049  * WiFi event handler
00050  ****************************************************************/
00051
00052 static void wifi_event_handler(void *arg, esp_event_base_t event_base,
00053                                int32_t event_id, void *event_data)
00054 {
00055     (void)arg;
00056     (void)event_data;
00057
00058     if ((event_base == WIFI_EVENT) && (event_id == WIFI_EVENT_STA_START)) {
00059         esp_wifi_connect();
00060     } else if ((event_base == WIFI_EVENT) &&
00061                (event_id == WIFI_EVENT_STA_DISCONNECTED)) {
00062         if (s_retry_num < WIFI_MAX_RETRY) {
00063             esp_wifi_connect();
00064             s_retry_num++;
00065             ESP_LOGW(TAG, "WiFi retry %d/%d", s_retry_num, WIFI_MAX_RETRY);
00066         } else {
00067             xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
00068         }
00069     } else if ((event_base == IP_EVENT) && (event_id == IP_EVENT_STA_GOT_IP)) {
00070         s_retry_num = 0;
00071         xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
00072     } else {
00073         /* other events ignored */
00074     }
00075 }
00076
00077 /****************************************************************
00078  * HTTP helper
00079  ****************************************************************/
00080
00081 /*
00082  * POST 'body' to s_rpc_url over HTTPS and read the response into
00083  * resp_buf (NUL-terminated on success).  Returns true if data was read.
00084  */
00085 static bool do_post(const char *body, char *resp_buf, size_t resp_buf_size)
00086 {
00087     bool success = false;
00088
00089     bool use_auth = ((s_project_id != NULL) && (s_project_id[0] != '\0') &&
00090                      (s_api_secret != NULL) && (s_api_secret[0] != '\0'));
00091
00092     esp_http_client_config_t cfg;
00093     (void)memset(&cfg, 0, sizeof(cfg));
00094     cfg.url               = s_rpc_url;
00095     cfg.method            = HTTP_METHOD_POST;
00096     cfg.timeout_ms        = 15000;
00097     cfg.crt_bundle_attach = esp_crt_bundle_attach;
00098     if (use_auth) {
00099         cfg.username  = s_project_id;
00100         cfg.password  = s_api_secret;
00101         cfg.auth_type = HTTP_AUTH_TYPE_BASIC;
00102     }
00103
00104     esp_http_client_handle_t client = esp_http_client_init(&cfg);
00105     if (client == NULL) {
00106         ESP_LOGE(TAG, "HTTP client init failed");
00107         return false;
00108     }
00109
00110     (void)esp_http_client_set_header(client, "Content-Type", "application/json");
00111
00112     int body_len = (int)strlen(body);
00113     esp_err_t err = esp_http_client_open(client, body_len);
00114     if (err != ESP_OK) {
00115         ESP_LOGE(TAG, "HTTP open: %s", esp_err_to_name(err));
00116         goto cleanup;
00117     }
00118
00119     if (esp_http_client_write(client, body, body_len) != body_len) {
00120         ESP_LOGE(TAG, "HTTP write incomplete");
00121         goto cleanup;
00122     }
00123
00124     {
00125         int64_t content_length = esp_http_client_fetch_headers(client);
00126         (void)content_length;  /* may be -1 for chunked; we read until EOF */
00127
00128         int total = 0;
```

```
00129         int read;
00130         do {
00131             int space = (int)(resp_buf_size - 1U) - total;
00132             if (space <= 0) { break; }
00133             read = esp_http_client_read(client, resp_buf + total, space);
00134             if (read > 0) { total += read; }
00135         } while (read > 0);
00136
00137         resp_buf[total] = '\0';
00138         success = (total > 0);
00139     }
00140
00141 cleanup:
00142     esp_http_client_close(client);
00143     esp_http_client_cleanup(client);
00144     return success;
00145 }
00146
00147 /*****************************************************************
00148  * Hex utilities
00149  *****************************************************************/
00150
00151 static char hex_nibble(uint8_t n)
00152 {
00153     return (n < 10U) ? (char)('0' + n) : (char)('a' + n - 10U);
00154 }
00155
00156 static void bytes_to_hex(const uint8_t *data, size_t len, char *out)
00157 {
00158     size_t i;
00159     for (i = 0U; i < len; i++) {
00160         out[i * HEX_PER_BYTE]       = hex_nibble((data[i] >> 4U) & 0x0FU);
00161         out[i * HEX_PER_BYTE + 1U]  = hex_nibble(data[i] & 0x0FU);
00162     }
00163 }
00164
00165 /*****************************************************************
00166  * Public API
00167  *****************************************************************/
00168
00169 void eth_rpc_init(const char *rpc_url, const char *from_addr)
00170 {
00171     s_rpc_url   = rpc_url;
00172     s_from_addr = from_addr;
00173 }
00174
00175 void eth_rpc_set_auth(const char *project_id, const char *api_secret)
00176 {
00177     s_project_id = project_id;
00178     s_api_secret = api_secret;
00179 }
00180
00181 bool eth_rpc_wifi_connect(const char *ssid, const char *password)
00182 {
00183     s_wifi_event_group = xEventGroupCreate();
00184     s_retry_num = 0;
00185
00186     ESP_ERROR_CHECK(esp_netif_init());
00187     ESP_ERROR_CHECK(esp_event_loop_create_default());
00188     (void)esp_netif_create_default_wifi_sta();
00189
00190     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
00191     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
00192
00193     esp_event_handler_instance_t h_any;
00194     esp_event_handler_instance_t h_ip;
00195     ESP_ERROR_CHECK(esp_event_handler_instance_register(
00196         WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler, NULL, &h_any));
00197     ESP_ERROR_CHECK(esp_event_handler_instance_register(
00198         IP_EVENT, IP_EVENT_STA_GOT_IP, &wifi_event_handler, NULL, &h_ip));
00199
00200     wifi_config_t wifi_cfg;
00201     (void)memset(&wifi_cfg, 0, sizeof(wifi_cfg));
00202     (void)strncpy((char *)wifi_cfg.sta.ssid,     ssid,     sizeof(wifi_cfg.sta.ssid)     - 1U);
00203     (void)strncpy((char *)wifi_cfg.sta.password, password, sizeof(wifi_cfg.sta.password) - 1U);
00204     wifi_cfg.sta.threshold.authmode = WIFI_AUTH_WPA2_PSK;
00205
00206     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
00207     ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_cfg));
00208     ESP_ERROR_CHECK(esp_wifi_start());
00209
00210     ESP_LOGI(TAG, "Connecting to \"%s\"...", ssid);
00211
00212     EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
00213                                            WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
00214                                            pdFALSE, pdFALSE,
00215                                            pdMS_TO_TICKS(WIFI_TIMEOUT_MS));
```

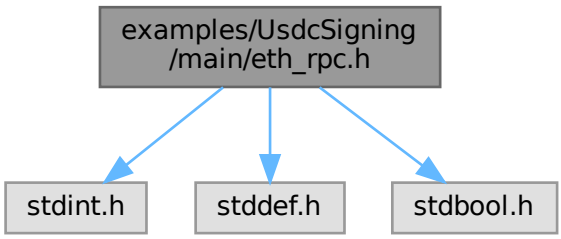
```
00216
00217     bool connected = ((bits & WIFI_CONNECTED_BIT) != 0U);
00218     if (connected) {
00219         ESP_LOGI(TAG, "WiFi connected");
00220     } else {
00221         ESP_LOGE(TAG, "WiFi connect failed");
00222     }
00223
00224     (void)esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP, h_ip);
00225     (void)esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID, h_any);
00226     vEventGroupDelete(s_wifi_event_group);
00227
00228     return connected;
00229 }
00230
00231 bool eth_rpc_get_nonce(uint64_t *nonce_out)
00232 {
00233     char body[256];
00234     (void)snprintf(body, sizeof(body),
00235                    "{\"jsonrpc\":\"2.0\",\"method\":\"eth_getTransactionCount\","
00236                    "\"params\":[\"%s\",\"pending\"],\"id\":1}",
00237                    s_from_addr);
00238
00239     char resp[RESP_BUF_SIZE];
00240     if (!do_post(body, resp, sizeof(resp))) {
00241         return false;
00242     }
00243
00244     /* Extract hex value after "result":"0x */
00245     char *result = strstr(resp, "\"result\":\"0x");
00246     if (result == NULL) {
00247         ESP_LOGE(TAG, "nonce: no result in: %s", resp);
00248         return false;
00249     }
00250     result += 12;  /* skip past "result":"0x */
00251
00252     *nonce_out = strtoull(result, NULL, 16);
00253     ESP_LOGI(TAG, "Nonce: %" PRIu64, *nonce_out);
00254     return true;
00255 }
00256
00257 uint8_t eth_rpc_ecrecover_parity(const uint8_t hash[32],
00258                                  const uint8_t r[32],
00259                                  const uint8_t s[32])
00260 {
00261     /* ecrecover precompile input: hash(32) || v_uint256(32) || r(32) || s(32) */
00262     uint8_t input[128];
00263     (void)memset(input, 0, sizeof(input));
00264     (void)memcpy(input, hash, 32U);
00265     /* v occupies the last byte of the second 32-byte slot (index 63) */
00266     (void)memcpy(input + 64U, r, 32U);
00267     (void)memcpy(input + 96U, s, 32U);
00268
00269     /* Hex-encode the 128-byte input */
00270     char input_hex[257];
00271     bytes_to_hex(input, sizeof(input), input_hex);
00272     input_hex[256] = '\0';
00273
00274     /* from_addr without "0x" prefix for comparison */
00275     const char *from_hex = s_from_addr;
00276     if ((from_hex[0] == '0') && ((from_hex[1] == 'x') || (from_hex[1] == 'X'))) {
00277         from_hex += 2;
00278     }
00279
00280     uint8_t v_raw;
00281     for (v_raw = 0U; v_raw < 2U; v_raw++) {
00282         /* Set v byte (27 or 28) in slot [32..63] last byte */
00283         input_hex[63U * HEX_PER_BYTE]      = hex_nibble(((27U + v_raw) >> 4U) & 0x0FU);
00284         input_hex[63U * HEX_PER_BYTE + 1U] = hex_nibble((27U + v_raw) & 0x0FU);
00285
00286         char body[600];
00287         (void)snprintf(body, sizeof(body),
00288                        "{\"jsonrpc\":\"2.0\",\"method\":\"eth_call\","
00289                        "\"params\":[{\"to\":"
00290                        "\"0x0000000000000000000000000000000000000001\","
00291                        "\"data\":\"0x%s\"},\"latest\"],\"id\":3}",
00292                        input_hex);
00293
00294         char resp[RESP_BUF_SIZE];
00295         if (!do_post(body, resp, sizeof(resp))) {
00296             continue;
00297         }
00298
00299         /* Response: "result":"0x" + 64 hex chars (32 bytes ABI address).
00300          * The address occupies the last 40 hex chars (bytes 12-31). */
00301         char *result = strstr(resp, "\"result\":\"0x");
00302         if (result == NULL) { continue; }
```

```
00303         result += 12U;  /* skip to hex digits */
00304
00305         size_t result_hex_len = 0U;
00306         {
00307             const char *p = result;
00308             while ((*p != '"') && (*p != '\0') && (*p != ',')) {
00309                 p++;
00310                 result_hex_len++;
00311             }
00312         }
00313
00314         if (result_hex_len < 64U) {
00315             /* Empty result -- address not recovered (try other v) */
00316             continue;
00317         }
00318
00319         /* ABI address: 24 hex chars of zeros + 40 hex chars of address */
00320         const char *recovered_hex = result + 24U;
00321
00322         if (strncasecmp(recovered_hex, from_hex, 40U) == 0) {
00323             ESP_LOGI(TAG, "v=%u matched ecrecover", v_raw);
00324             return v_raw;
00325         }
00326     }
00327
00328     ESP_LOGW(TAG, "ecrecover did not match either parity, defaulting v=0");
00329     return 0U;
00330 }
00331
00332 bool eth_rpc_send_raw_tx(const uint8_t *tx, size_t tx_len,
00333                          char *tx_hash_out, size_t tx_hash_max)
00334 {
00335     /* "0x" + 2 hex chars per byte + NUL */
00336     size_t hex_str_size = 2U + tx_len * HEX_PER_BYTE + 1U;
00337     char *tx_hex = (char *)malloc(hex_str_size);
00338     if (tx_hex == NULL) { return false; }
00339
00340     tx_hex[0] = '0';
00341     tx_hex[1] = 'x';
00342     bytes_to_hex(tx, tx_len, tx_hex + 2U);
00343     tx_hex[hex_str_size - 1U] = '\0';
00344
00345     /* JSON body */
00346     size_t body_size = hex_str_size + 128U;
00347     char *body = (char *)malloc(body_size);
00348     if (body == NULL) { free(tx_hex); return false; }
00349
00350     (void)snprintf(body, body_size,
00351                    "{\"jsonrpc\":\"2.0\",\"method\":\"eth_sendRawTransaction\","
00352                    "\"params\":[\"%s\"],\"id\":2}",
00353                    tx_hex);
00354     free(tx_hex);
00355
00356     char resp[RESP_BUF_SIZE];
00357     bool ok = do_post(body, resp, sizeof(resp));
00358     free(body);
00359
00360     if (!ok) { return false; }
00361
00362     /* Extract "result":"0x..." (the tx hash) */
00363     char *result = strstr(resp, "\"result\":\"");
00364     if (result == NULL) {
00365         ESP_LOGE(TAG, "send_raw_tx: no result in: %s", resp);
00366         return false;
00367     }
00368     result += 10U;  /* skip past "result":" -- now at '0x...' */
00369
00370     char *end = strchr(result, '"');
00371     if (end == NULL) { return false; }
00372
00373     size_t hash_len = (size_t)(end - result);
00374     if ((hash_len + 1U) > tx_hash_max) { return false; }
00375
00376     (void)memcpy(tx_hash_out, result, hash_len);
00377     tx_hash_out[hash_len] = '\0';
00378     return true;
00379 }
```

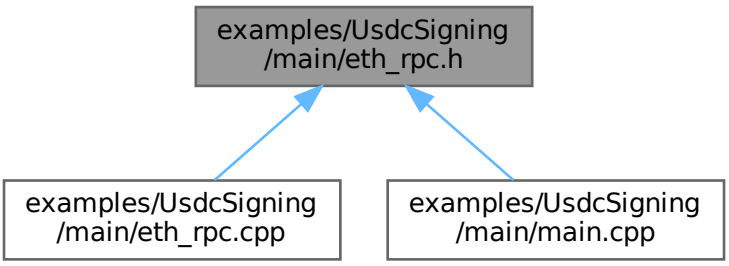
5.48 examples/UsdcSigning/main/eth_rpc.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include <stdbool.h>
```

Include dependency graph for eth_rpc.h:



This graph shows which files directly or indirectly include this file:



Functions

- void eth_rpc_init (const char ∗rpc_url, const char ∗from_addr)
- void eth_rpc_set_auth (const char ∗project_id, const char ∗api_secret)
- bool eth_rpc_wifi_connect (const char ∗ssid, const char ∗password)
- bool eth_rpc_get_nonce (uint64_t ∗nonce_out)
- uint8_t eth_rpc_ecrecover_parity (const uint8_t hash[32], const uint8_t r[32], const uint8_t s[32])
- bool eth_rpc_send_raw_tx (const uint8_t ∗tx, size_t tx_len, char ∗tx_hash_out, size_t tx_hash_↩ max)

5.48.1 Function Documentation

5.48.1.1 eth_rpc_ecrecover_parity()

```
uint8_t eth_rpc_ecrecover_parity (
            const uint8_t hash[32],
            const uint8_t r[32],
            const uint8_t s[32])
```

Definition at line 257 of file eth_rpc.cpp.
References bytes_to_hex(), do_post(), hex_nibble(), HEX_PER_BYTE, RESP_BUF_SIZE, s_from_addr, and TAG.
Referenced by signing_loop().

5.48.1.2 eth_rpc_get_nonce()

```
bool eth_rpc_get_nonce (
            uint64_t * nonce_out)
```

Definition at line 231 of file eth_rpc.cpp.
References do_post(), RESP_BUF_SIZE, s_from_addr, and TAG.
Referenced by signing_loop().

5.48.1.3 eth_rpc_init()

```
void eth_rpc_init (
            const char * rpc_url,
            const char * from_addr)
```

Definition at line 169 of file eth_rpc.cpp.
References s_from_addr, and s_rpc_url.
Referenced by app_main().

5.48.1.4 eth_rpc_send_raw_tx()

```
bool eth_rpc_send_raw_tx (
            const uint8_t * tx,
            size_t tx_len,
            char * tx_hash_out,
            size_t tx_hash_max)
```

Definition at line 332 of file eth_rpc.cpp.
References bytes_to_hex(), do_post(), HEX_PER_BYTE, RESP_BUF_SIZE, and TAG.
Referenced by signing_loop().

5.48.1.5 eth_rpc_set_auth()

```
void eth_rpc_set_auth (
            const char * project_id,
            const char * api_secret)
```

Definition at line 175 of file eth_rpc.cpp.
References s_api_secret, and s_project_id.
Referenced by app_main().

5.48.1.6 eth_rpc_wifi_connect()

```
bool eth_rpc_wifi_connect (
            const char * ssid,
            const char * password)
```

Definition at line 181 of file eth_rpc.cpp.
References s_retry_num, s_wifi_event_group, TAG, WIFI_CONNECTED_BIT, wifi_event_handler(), WIFI_FAIL_BIT, and WIFI_TIMEOUT_MS.
Referenced by app_main().

5.49 eth_rpc.h

Go to the documentation of this file.

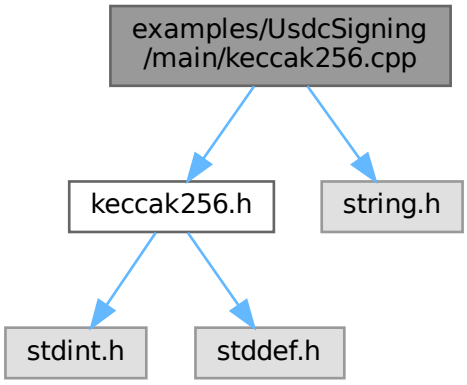
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #pragma once
00007
00008 #include <stdint.h>
00009 #include <stddef.h>
00010 #include <stdbool.h>
00011
00012 #ifdef __cplusplus
00013 extern "C" {
00014 #endif
```

```
00015
00016 /*
00017  * Set the RPC URL and the from-address used for nonce queries and
00018  * ecrecover comparison.  Must be called before any other eth_rpc_* function.
00019  * from_addr must be a "0x..."-prefixed 40-hex-char string.
00020  */
00021 void eth_rpc_init(const char *rpc_url, const char *from_addr);
00022
00023 /*
00024  * Optional: set the Infura API secret for HTTP Basic Auth.
00025  * Pass NULL or empty string to disable.  Must be called before any
00026  * eth_rpc_* function that makes network requests.
00027  * The Project ID is used as the username; api_secret is the password.
00028  */
00029 void eth_rpc_set_auth(const char *project_id, const char *api_secret);
00030
00031 /*
00032  * Connect to a WiFi network and block until an IP is obtained (up to 30 s).
00033  * Returns true on success.
00034  */
00035 bool eth_rpc_wifi_connect(const char *ssid, const char *password);
00036
00037 /*
00038  * Fetch the pending transaction count (nonce) for from_addr.
00039  * Returns true and writes the nonce into *nonce_out on success.
00040  */
00041 bool eth_rpc_get_nonce(uint64_t *nonce_out);
00042
00043 /*
00044  * Determine the signature parity bit (v = 0 or 1) by calling the ecrecover
00045  * precompile (address 0x01) via eth_call and matching the recovered address
00046  * against from_addr.  Returns 0 or 1; defaults to 0 on RPC failure.
00047  */
00048 uint8_t eth_rpc_ecrecover_parity(const uint8_t hash[32],
00049                                  const uint8_t r[32],
00050                                  const uint8_t s[32]);
00051
00052 /*
00053  * Broadcast a raw signed transaction (type-prefixed RLP bytes).
00054  * On success writes the "0x..."-prefixed tx hash into tx_hash_out and
00055  * returns true.  tx_hash_out must be at least 68 bytes (2 + 64 + NUL).
00056  */
00057 bool eth_rpc_send_raw_tx(const uint8_t *tx, size_t tx_len,
00058                          char *tx_hash_out, size_t tx_hash_max);
00059
00060 #ifdef __cplusplus
00061 }
00062 #endif
```

5.50 examples/UsdcSigning/main/keccak256.cpp File Reference

```
#include "keccak256.h"
#include <string.h>
```

Include dependency graph for keccak256.cpp:



Macros

- #define KECCAK_ROUNDS 24U
- #define KECCAK_RATE 136U /* 1088 / 8 bytes — rate for 256-bit output */
- #define KECCAK_STATE_LANE 25U /* 5×5 uint64_t lanes */

Functions

- static uint64_t rot64 (uint64_t x, uint8_t n)
- static void keccak_f1600 (uint64_t st[KECCAK_STATE_LANE])
- void keccak256 (const uint8_t *input, size_t length, uint8_t digest[32])

Variables

- static const uint64_t kRC [KECCAK_ROUNDS]
- static const uint8_t kRHO [KECCAK_STATE_LANE]

5.50.1 Macro Definition Documentation

5.50.1.1 KECCAK_RATE

```
#define KECCAK_RATE 136U /* 1088 / 8 bytes -- rate for 256-bit output */
```

Definition at line 10 of file keccak256.cpp.
Referenced by keccak256().

5.50.1.2 KECCAK_ROUNDS

```
#define KECCAK_ROUNDS 24U
```

Definition at line 9 of file keccak256.cpp.
Referenced by keccak_f1600().

5.50.1.3 KECCAK_STATE_LANE

```
#define KECCAK_STATE_LANE 25U /* 5×5 uint64_t lanes */
```

Definition at line 11 of file keccak256.cpp.
Referenced by keccak256(), and keccak_f1600().

5.50.2 Function Documentation

5.50.2.1 keccak256()

```
void keccak256 (
            const uint8_t * input,
            size_t length,
            uint8_t digest[32])
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 85 of file keccak256.cpp.
References keccak_f1600(), KECCAK_RATE, and KECCAK_STATE_LANE.
Referenced by signing_loop().

5.50.2.2 keccak_f1600()

```
void keccak_f1600 (
            uint64_t st[KECCAK_STATE_LANE])  [static]
```

Definition at line 42 of file keccak256.cpp.
References KECCAK_ROUNDS, KECCAK_STATE_LANE, kRC, kRHO, and rot64().
Referenced by keccak256().

5.50.2.3 rot64()

```
uint64_t rot64 (
            uint64_t x,
            uint8_t n)  [static]
```

Definition at line 37 of file keccak256.cpp.
Referenced by keccak_f1600().

5.50.3 Variable Documentation

5.50.3.1 kRC

```
const uint64_t kRC[KECCAK_ROUNDS]  [static]
```

Initial value:

```
= {
    0x0000000000000001ULL, 0x0000000000008082ULL,
    0x800000000000808AULL, 0x8000000080008000ULL,
    0x000000000000808BULL, 0x0000000080000001ULL,
    0x8000000080008081ULL, 0x8000000000008009ULL,
    0x000000000000008AULL, 0x0000000000000088ULL,
    0x0000000080008009ULL, 0x000000008000000AULL,
    0x000000008000808BULL, 0x800000000000008BULL,
    0x8000000000008089ULL, 0x8000000000008003ULL,
    0x8000000000008002ULL, 0x8000000000000080ULL,
    0x000000000000800AULL, 0x800000008000000AULL,
    0x8000000080008081ULL, 0x8000000000008080ULL,
    0x0000000080000001ULL, 0x8000000080008008ULL,
}
```

Definition at line 13 of file keccak256.cpp.
Referenced by keccak_f1600().

5.50.3.2 kRHO

```
const uint8_t kRHO[KECCAK_STATE_LANE]  [static]
```

Initial value:

```
= {
     0,  1, 62, 28, 27,
    36, 44,  6, 55, 20,
     3, 10, 43, 25, 39,
    41, 45, 15, 21,  8,
    18,  2, 61, 56, 14
}
```

Definition at line 29 of file keccak256.cpp.

Referenced by keccak_f1600().

5.51 keccak256.cpp

Go to the documentation of this file.

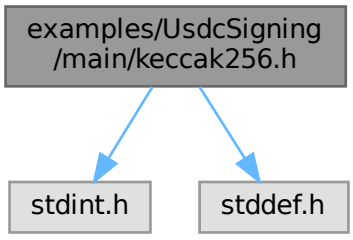
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #include "keccak256.h"
00007 #include <string.h>
00008
00009 #define KECCAK_ROUNDS     24U
00010 #define KECCAK_RATE       136U   /* 1088 / 8 bytes -- rate for 256-bit output */
00011 #define KECCAK_STATE_LANE 25U    /* 5×5 uint64_t lanes                        */
00012
00013 static const uint64_t kRC[KECCAK_ROUNDS] = {
00014     0x0000000000000001ULL, 0x0000000000008082ULL,
00015     0x800000000000808AULL, 0x8000000080008000ULL,
00016     0x000000000000808BULL, 0x0000000080000001ULL,
00017     0x8000000080008081ULL, 0x8000000000008009ULL,
00018     0x000000000000008AULL, 0x0000000000000088ULL,
00019     0x0000000080008009ULL, 0x000000008000000AULL,
00020     0x000000008000808BULL, 0x800000000000008BULL,
00021     0x8000000000008089ULL, 0x8000000000008003ULL,
00022     0x8000000000008002ULL, 0x8000000000000080ULL,
00023     0x000000000000800AULL, 0x800000008000000AULL,
00024     0x8000000080008081ULL, 0x8000000000008080ULL,
00025     0x0000000080000001ULL, 0x8000000080008008ULL,
00026 };
00027
00028 /* Rho rotation offsets for lane [x + 5*y], derived from the Keccak spec. */
00029 static const uint8_t kRHO[KECCAK_STATE_LANE] = {
00030      0,  1, 62, 28, 27,   /* y=0 */
00031     36, 44,  6, 55, 20,   /* y=1 */
00032      3, 10, 43, 25, 39,   /* y=2 */
00033     41, 45, 15, 21,  8,   /* y=3 */
00034     18,  2, 61, 56, 14    /* y=4 */
00035 };
00036
00037 static uint64_t rot64(uint64_t x, uint8_t n)
00038 {
00039     return (n == 0U) ? x : ((x << n) | (x >> (64U - n)));
00040 }
00041
00042 static void keccak_f1600(uint64_t st[KECCAK_STATE_LANE])
00043 {
00044     uint64_t C[5], D[5], B[KECCAK_STATE_LANE];
00045     unsigned int round, x, y;
00046
00047     for (round = 0U; round < KECCAK_ROUNDS; round++) {
00048         /* Theta */
00049         for (x = 0U; x < 5U; x++) {
00050             C[x] = st[x] ^ st[x + 5U] ^ st[x + 10U] ^ st[x + 15U] ^ st[x + 20U];
00051         }
00052         for (x = 0U; x < 5U; x++) {
00053             D[x] = C[(x + 4U) % 5U] ^ rot64(C[(x + 1U) % 5U], 1U);
00054         }
00055         for (x = 0U; x < 5U; x++) {
00056             for (y = 0U; y < 5U; y++) {
00057                 st[x + 5U * y] ^= D[x];
00058             }
00059         }
00060
00061         /* Rho + Pi: B[dst] = ROT(st[src], rho[src]) */
00062         for (x = 0U; x < 5U; x++) {
00063             for (y = 0U; y < 5U; y++) {
00064                 unsigned int src   = x + 5U * y;
00065                 unsigned int dst_x = y;
00066                 unsigned int dst_y = (2U * x + 3U * y) % 5U;
00067                 B[dst_x + 5U * dst_y] = rot64(st[src], kRHO[src]);
00068             }
00069         }
00070
00071         /* Chi */
00072         for (x = 0U; x < 5U; x++) {
00073             for (y = 0U; y < 5U; y++) {
00074                 st[x + 5U * y] = B[x + 5U * y] ^
00075                                  ((~B[(x + 1U) % 5U + 5U * y]) &
00076                                     B[(x + 2U) % 5U + 5U * y]);
00077             }
00078         }
```

```
00079
00080         /* Iota */
00081         st[0] ^= kRC[round];
00082     }
00083 }
00084
00085 void keccak256(const uint8_t *input, size_t length, uint8_t digest[32])
00086 {
00087     uint64_t state[KECCAK_STATE_LANE];
00088     uint8_t *sb = (uint8_t *)state;
00089     size_t offset = 0U;
00090     size_t i;
00091
00092     (void)memset(state, 0, sizeof(state));
00093
00094     /* Absorb full blocks */
00095     while ((length - offset) >= KECCAK_RATE) {
00096         for (i = 0U; i < KECCAK_RATE; i++) {
00097             sb[i] ^= input[offset + i];
00098         }
00099         keccak_f1600(state);
00100         offset += KECCAK_RATE;
00101     }
00102
00103     /* Absorb remaining bytes */
00104     size_t rem = length - offset;
00105     for (i = 0U; i < rem; i++) {
00106         sb[i] ^= input[offset + i];
00107     }
00108
00109     /* Pad: 0x01 for Ethereum Keccak (pre-NIST), 0x80 at end of rate block */
00110     sb[rem]                 ^= 0x01U;
00111     sb[KECCAK_RATE - 1U] ^= 0x80U;
00112
00113     keccak_f1600(state);
00114
00115     /* Squeeze first 32 bytes */
00116     (void)memcpy(digest, sb, 32U);
00117 }
```

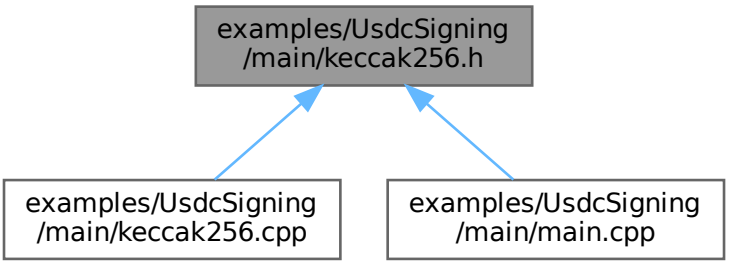
5.52 examples/UsdcSigning/main/keccak256.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for keccak256.h:



This graph shows which files directly or indirectly include this file:



Functions

- void keccak256 (const uint8_t *input, size_t length, uint8_t digest[32])

5.52.1 Function Documentation

5.52.1.1 keccak256()

```
void keccak256 (
            const uint8_t * input,
            size_t length,
            uint8_t digest[32])
```

Definition at line 85 of file keccak256.cpp.
References keccak_f1600(), KECCAK_RATE, and KECCAK_STATE_LANE.
Referenced by signing_loop().

5.53 keccak256.h

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #pragma once
00007
00008 #include <stdint.h>
00009 #include <stddef.h>
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014
00015 /* Ethereum uses the original Keccak-256 (0x01 padding), NOT SHA3-256 (0x06). */
00016 void keccak256(const uint8_t *input, size_t length, uint8_t digest[32]);
00017
00018 #ifdef __cplusplus
00019 }
00020 #endif
```

5.54 examples/BasicUsage/config.template.h File Reference

Macros

- #define WIFI_SSID "<YOUR_SSID>"

- #define WIFI_PASSWORD "<YOUR_PASSWORD>"

5.54.1 Macro Definition Documentation

5.54.1.1 WIFI_PASSWORD

```
#define WIFI_PASSWORD "<YOUR_PASSWORD>"
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 13 of file config.template.h.
Referenced by app_main(), and wifi_start().

5.54.1.2 WIFI_SSID

```
#define WIFI_SSID "<YOUR_SSID>"
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 12 of file config.template.h.
Referenced by app_main(), and wifi_start().

5.55 config.template.h

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #ifndef CONFIG_H
00007 #define CONFIG_H
00008
00009 /* - WiFi credentials ------------------------------
00010  * WiFi is started on boot so the radio feeds the hardware TRNG with
00011  * full entropy.  Fill in the SSID and password for your network. */
00012 #define WIFI_SSID       "<YOUR_SSID>"
00013 #define WIFI_PASSWORD   "<YOUR_PASSWORD>"
00014
00015 #endif /* CONFIG_H */
```

5.56 examples/Connect/config.template.h File Reference

Macros

- #define WIFI_SSID "<YOUR_SSID>"
- #define WIFI_PASSWORD "<YOUR_PASSWORD>"

5.56.1 Macro Definition Documentation

5.56.1.1 WIFI_PASSWORD

```
#define WIFI_PASSWORD "<YOUR_PASSWORD>"
```

Definition at line 13 of file config.template.h.

5.56.1.2 WIFI_SSID

```
#define WIFI_SSID "<YOUR_SSID>"
```
Definition at line 12 of file config.template.h.

5.57 config.template.h

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #ifndef CONFIG_H
00007 #define CONFIG_H
00008
00009 /* - WiFi credentials ------------------------------
00010  * WiFi is started on boot so the radio feeds the hardware TRNG with
00011  * full entropy.  Fill in the SSID and password for your network. */
00012 #define WIFI_SSID       "<YOUR_SSID>"
00013 #define WIFI_PASSWORD   "<YOUR_PASSWORD>"
00014
00015 #endif /* CONFIG_H */
```

5.58 examples/Sign/config.template.h File Reference

Macros

- #define WIFI_SSID "<YOUR_SSID>"
- #define WIFI_PASSWORD "<YOUR_PASSWORD>"

5.58.1 Macro Definition Documentation

5.58.1.1 WIFI_PASSWORD

```
#define WIFI_PASSWORD "<YOUR_PASSWORD>"
```
Definition at line 13 of file config.template.h.

5.58.1.2 WIFI_SSID

```
#define WIFI_SSID "<YOUR_SSID>"
```
Definition at line 12 of file config.template.h.

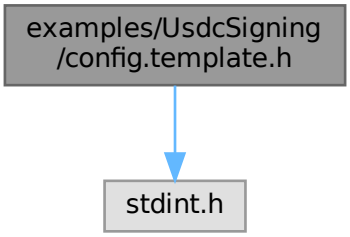
5.59 config.template.h

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #ifndef CONFIG_H
00007 #define CONFIG_H
00008
00009 /* - WiFi credentials ------------------------------
00010  * WiFi is started on boot so the radio feeds the hardware TRNG with
00011  * full entropy.  Fill in the SSID and password for your network. */
00012 #define WIFI_SSID       "<YOUR_SSID>"
00013 #define WIFI_PASSWORD   "<YOUR_PASSWORD>"
00014
00015 #endif /* CONFIG_H */
```

5.60 examples/UsdcSigning/config.template.h File Reference

`#include <stdint.h>`
Include dependency graph for config.template.h:



Macros

- #define WIFI_SSID "<YOUR_SSID>"
- #define WIFI_PASSWORD "<YOUR_PASSWORD>"
- #define RPC_HOST "ethereum-sepolia-rpc.publicnode.com"
- #define RPC_PORT 443
- #define RPC_URL "https://" RPC_HOST
- #define CARD_PIN "<CARD_PIN>" /* 4-9 digit PIN, e.g. "000000000" */
- #define CARD_PIN_LEN (9U) /* number of digits in CARD_PIN */
- #define ADDR_FROM "<SENDER_ADDRESS>"
- #define ADDR_TO "<RECIPIENT_ADDRESS>"
- #define ADDR_USDC "<USDC_CONTRACT_ADDRESS>"
- #define CHAIN_ID_SEPOLIA 11155111
- #define AMOUNT_USDC 1000000UL /* 1.0 USDC */
- #define MAX_PRIORITY_FEE 2000000000ULL /* 2 Gwei */
- #define MAX_FEE 4000000000ULL /* 4 Gwei */
- #define GAS_LIMIT_ERC20 60000ULL

5.60.1 Macro Definition Documentation

5.60.1.1 ADDR_FROM

`#define ADDR_FROM "<SENDER_ADDRESS>"`

Examples

UsdcSigning/main/main.cpp.

Definition at line 62 of file config.template.h.
Referenced by app_main().

5.60.1.2 ADDR_TO

`#define ADDR_TO "<RECIPIENT_ADDRESS>"`

Examples

UsdcSigning/main/main.cpp.

Definition at line 65 of file config.template.h.
Referenced by signing_loop().

5.60.1.3 ADDR_USDC

```
#define ADDR_USDC "<USDC_CONTRACT_ADDRESS>"
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 68 of file config.template.h.
Referenced by signing_loop().

5.60.1.4 AMOUNT_USDC

```
#define AMOUNT_USDC 1000000UL /* 1.0 USDC */
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 76 of file config.template.h.
Referenced by signing_loop().

5.60.1.5 CARD_PIN

```
#define CARD_PIN "<CARD_PIN>" /* 4-9 digit PIN, e.g.  "000000000" */
```

NEVER COMMIT config.h — it contains credentials. Add config.h to .gitignore.

Examples

UsdcSigning/main/main.cpp.

Definition at line 55 of file config.template.h.
Referenced by signing_loop().

5.60.1.6 CARD_PIN_LEN

```
#define CARD_PIN_LEN (9U) /* number of digits in CARD_PIN */
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 56 of file config.template.h.
Referenced by signing_loop().

5.60.1.7 CHAIN_ID_SEPOLIA

```
#define CHAIN_ID_SEPOLIA 11155111
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 73 of file config.template.h.
Referenced by signing_loop().

5.60.1.8 GAS_LIMIT_ERC20

```
#define GAS_LIMIT_ERC20 60000ULL
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 81 of file config.template.h.
Referenced by signing_loop().

5.60.1.9 MAX_FEE

```
#define MAX_FEE 4000000000ULL /* 4 Gwei */
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 80 of file config.template.h.
Referenced by signing_loop().

5.60.1.10 MAX_PRIORITY_FEE

```
#define MAX_PRIORITY_FEE 2000000000ULL /* 2 Gwei */
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 79 of file config.template.h.
Referenced by signing_loop().

5.60.1.11 RPC_HOST

```
#define RPC_HOST "ethereum-sepolia-rpc.publicnode.com"
```

Choose ONE provider below and comment out the other.
Option A — PublicNode (free, no account required) Uncomment the three lines under "Option A".
Option B — Infura (requires a free account at app.infura.io)

1. Create an API key in the Infura dashboard.
2. In the key's Settings tab, reveal (or generate) the API Secret.
3. Uncomment the four lines under "Option B" and fill in the values. Note: the API Secret must have NO leading or trailing spaces.

Definition at line 34 of file config.template.h.

5.60.1.12 RPC_PORT

```
#define RPC_PORT 443
```

Definition at line 35 of file config.template.h.

5.60.1.13 RPC_URL

```
#define RPC_URL "https://" RPC_HOST
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 36 of file config.template.h.
Referenced by app_main().

5.60.1.14 WIFI_PASSWORD

```
#define WIFI_PASSWORD "<YOUR_PASSWORD>"
```

Definition at line 15 of file config.template.h.

5.60.1.15 WIFI_SSID

```
#define WIFI_SSID "<YOUR_SSID>"
```

Definition at line 14 of file config.template.h.

5.61 config.template.h

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #ifndef CONFIG_H
00007 #define CONFIG_H
00008
00009 #include <stdint.h>
00010
00011 /* ========================
00012  * WiFi Configuration
00013  * ======================== */
00014 #define WIFI_SSID         "<YOUR_SSID>"
00015 #define WIFI_PASSWORD     "<YOUR_PASSWORD>"
00016
00017 /* ========================
00018  * Ethereum / RPC
00019  * ======================== */
00032
00033 /* --- Option A: PublicNode ---------------------------------------- */
00034 #define RPC_HOST        "ethereum-sepolia-rpc.publicnode.com"
00035 #define RPC_PORT        443
00036 #define RPC_URL         "https://" RPC_HOST
00037 /* No authentication needed -- leave RPC_PROJECT_ID / RPC_API_SECRET
00038  * undefined (or comment them out) when using PublicNode.           */
00039
00040 /* --- Option B: Infura -------------------------------------------- */
00041 /* #define RPC_HOST         "sepolia.infura.io"                      */
00042 /* #define RPC_PORT         443                                       */
00043 /* #define RPC_PROJECT_ID   "<YOUR_INFURA_PROJECT_ID>"               */
00044 /* #define RPC_URL          "https://sepolia.infura.io/v3/" RPC_PROJECT_ID */
00045 /* #define RPC_API_SECRET   "<YOUR_INFURA_API_SECRET>"               */
00046
00047 /* ========================
00048  * Wallet / Keys (SENSITIVE)
00049  * ======================== */
00054
00055 #define CARD_PIN      "<CARD_PIN>"    /* 4-9 digit PIN, e.g. "000000000" */
00056 #define CARD_PIN_LEN  (9U)            /* number of digits in CARD_PIN     */
00057
00058 /* ========================
00059  * Ethereum Addresses
00060  * ======================== */
00061 /* Sender address -- lowercase hex, no 0x prefix */
00062 #define ADDR_FROM         "<SENDER_ADDRESS>"
00063
00064 /* Recipient address */
00065 #define ADDR_TO           "<RECIPIENT_ADDRESS>"
00066
00067 /* USDC contract address (Sepolia testnet) */
00068 #define ADDR_USDC         "<USDC_CONTRACT_ADDRESS>"
00069
00070 /* ========================
00071  * Transaction Parameters
00072  * ======================== */
00073 #define CHAIN_ID_SEPOLIA  11155111
00074
00075 /* Amount in token smallest unit (USDC has 6 decimals) */
00076 #define AMOUNT_USDC       1000000UL   /* 1.0 USDC */
00077
00078 /* Gas parameters (in wei) */
00079 #define MAX_PRIORITY_FEE  2000000000ULL  /* 2 Gwei */
00080 #define MAX_FEE           4000000000ULL  /* 4 Gwei */
00081 #define GAS_LIMIT_ERC20   60000ULL
00082
00083 #endif /* CONFIG_H */
```

5.62 examples/VerifyPin/config.template.h File Reference

Macros

- #define WIFI_SSID "<YOUR_SSID>"
- #define WIFI_PASSWORD "<YOUR_PASSWORD>"

5.62.1 Macro Definition Documentation

5.62.1.1 WIFI_PASSWORD

```
#define WIFI_PASSWORD "<YOUR_PASSWORD>"
```

Definition at line 13 of file config.template.h.

5.62.1.2 WIFI_SSID

```
#define WIFI_SSID "<YOUR_SSID>"
```

Definition at line 12 of file config.template.h.

5.63 config.template.h

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #ifndef CONFIG_H
00007 #define CONFIG_H
00008
00009 /* - WiFi credentials ------------------------------
00010  * WiFi is started on boot so the radio feeds the hardware TRNG with
00011  * full entropy.  Fill in the SSID and password for your network. */
00012 #define WIFI_SSID       "<YOUR_SSID>"
00013 #define WIFI_PASSWORD   "<YOUR_PASSWORD>"
00014
00015 #endif /* CONFIG_H */
```

5.64 examples/BasicUsage/main/main.cpp File Reference

```
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "driver/spi_master.h"
#include "esp_log.h"
#include "esp_wifi.h"
#include "esp_netif.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "pn532.h"
#include "CryptnoxWallet.h"
#include "Pn532NfcTransport.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "ESP32Platform.h"
#include "CW_Utils.h"
#include "config.h"
```

Include dependency graph for main.cpp:

Macros

- #define SPI_ENABLED 1
- #define I2C_ENABLED 0
- #define SPI_MOSI 11
- #define SPI_MISO 13
- #define SPI_SCLK 12
- #define SPI_MAX_TRANSFER_SZ 4096
- #define SPI_PIN_UNUSED (-1)
- #define NFC_CS 10
- #define WIFI_CONNECTED_BIT BIT0
- #define WIFI_FAIL_BIT BIT1

Functions

- static void wifi_event_handler (void *arg, esp_event_base_t event_base, int32_t event_id, void *event_data)

  FreeRTOS event handler driving the Wi-Fi station state machine.
- static void wifi_start (void)

  Initialise Wi-Fi station mode and block until connected or timeout.
- static void run_basic_usage_loop (CryptnoxWallet &wallet)

  Main application loop: connect, sign a test hash, wipe buffers, disconnect.
- void app_main (void)

  ESP-IDF application entry point.

Variables

- static const char *const TAG = "basic_usage"
- static const uint32_t LOOP_DELAY_MS = 1000U
- static const uint32_t WIFI_TIMEOUT_MS = 10000U
- static const int WIFI_MAX_RETRY = 5
- static EventGroupHandle_t s_wifi_event_group
- static int s_retry_num = 0
- static const uint8_t DEFAULT_PIN [ ] = "000000000"

  Demo PIN — replace with the PIN used during card initialisation.
- static const size_t DEFAULT_PIN_LEN = sizeof(DEFAULT_PIN) - 1U

5.64.1 Macro Definition Documentation

5.64.1.1 I2C_ENABLED

```
#define I2C_ENABLED 0
```

Definition at line 50 of file main.cpp.

5.64.1.2 NFC_CS

```
#define NFC_CS 10
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 59 of file main.cpp.
Referenced by app_main().

5.64.1.3 SPI_ENABLED

```
#define SPI_ENABLED 1
```

Definition at line 49 of file main.cpp.

5.64.1.4 SPI_MAX_TRANSFER_SZ

```
#define SPI_MAX_TRANSFER_SZ 4096
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 57 of file main.cpp.
Referenced by app_main().

5.64.1.5 SPI_MISO

```
#define SPI_MISO 13
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 55 of file main.cpp.
Referenced by app_main().

5.64.1.6 SPI_MOSI

```
#define SPI_MOSI 11
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 54 of file main.cpp.
Referenced by app_main().

5.64.1.7 SPI_PIN_UNUSED

```
#define SPI_PIN_UNUSED (-1)
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 58 of file main.cpp.
Referenced by app_main().

5.64.1.8 SPI_SCLK

```
#define SPI_SCLK 12
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 56 of file main.cpp.
Referenced by app_main().

5.64.1.9 WIFI_CONNECTED_BIT

```
#define WIFI_CONNECTED_BIT BIT0
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 77 of file main.cpp.
Referenced by eth_rpc_wifi_connect(), wifi_event_handler(), wifi_event_handler(), and wifi_start().

5.64.1.10 WIFI_FAIL_BIT

```
#define WIFI_FAIL_BIT BIT1
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 78 of file main.cpp.
Referenced by eth_rpc_wifi_connect(), wifi_event_handler(), wifi_event_handler(), and wifi_start().

5.64.2 Function Documentation

5.64.2.1 app_main()

```
void app_main (
            void )
```

ESP-IDF application entry point.
Initialises NVS, starts Wi-Fi for full TRNG entropy, brings up the SPI or I^2C bus and PN532 reader, then enters run_basic_usage_loop.

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, UsdcSigning/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 284 of file main.cpp.
References ESP32Logger::begin(), pn532_config_t::i2c_clock_hz, pn532_config_t::i2c_port, NFC_CS, pn532_config_t::pin_cs, pn532_config_t::pin_irq, pn532_config_t::pin_rst, pn532_config_t::pin_scl, pn532_config_t::pin_sda, pn532_init(), PN532_TRANSPORT_I2C, PN532_TRANSPORT_SPI, Pn532NfcTransport::printFirmwareVersion(), run_basic_usage_loop(), pn532_config_t::skip_bus_init, pn532_config_t::spi_host, SPI_MAX_TRANSFER_SZ, SPI_MISO, SPI_MOSI, SPI_PIN_UNUSED, SPI_SCLK, TAG, pn532_config_t::transport, and wifi_start().

5.64.2.2 run_basic_usage_loop()

```
void run_basic_usage_loop (
            CryptnoxWallet & wallet)  [static]
```

Main application loop: connect, sign a test hash, wipe buffers, disconnect.
Each iteration establishes the secure channel, signs a 32-byte test hash, prints the raw rs bytes, securely wipes sensitive buffers, then disconnects. Halts permanently on CW_SIGN_PIN_INCORRECT to protect the card's retry counter.

Parameters

in	*wallet*	Initialised wallet instance.

Examples

BasicUsage/main/main.cpp.

Definition at line 192 of file main.cpp.
References DEFAULT_PIN, DEFAULT_PIN_LEN, LOOP_DELAY_MS, and TAG.
Referenced by app_main().

5.64.2.3 wifi_event_handler()

```
void wifi_event_handler (
            void * arg,
            esp_event_base_t event_base,
            int32_t event_id,
            void * event_data)  [static]
```

FreeRTOS event handler driving the Wi-Fi station state machine.
Handles `WIFI_EVENT_STA_START` (triggers connection), `WIFI_EVENT_STA_DISCONNECTED` (retries up to `WIFI_MAX_RETRY` times), and `IP_EVENT_STA_GOT_IP` (signals success via the event group).

Parameters

`in`	*arg*	Unused.
`in`	*event_base*	Event base (`WIFI_EVENT` or `IP_EVENT`).
`in`	*event_id*	Event identifier within `event_base`.
`in`	*event_data*	Event-specific data (unused).

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 95 of file main.cpp.
References s_retry_num, s_wifi_event_group, WIFI_CONNECTED_BIT, WIFI_FAIL_BIT, and WIFI_MAX_RETRY.
Referenced by eth_rpc_wifi_connect(), and wifi_start().

5.64.2.4 wifi_start()

```
void wifi_start (
            void )  [static]
```

Initialise Wi-Fi station mode and block until connected or timeout.
Starts the ESP Wi-Fi stack, registers wifi_event_handler, configures the SSID and password from WIFI_SSID / WIFI_PASSWORD in `config.h`, then waits up to `WIFI_TIMEOUT_MS` for an IP address. The radio must be active before crypto operations so the hardware TRNG operates with full entropy (SEC-001).

Note

If the connection fails the function logs a warning and returns normally; the firmware continues with reduced TRNG entropy.

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 130 of file main.cpp.
References s_retry_num, s_wifi_event_group, TAG, WIFI_CONNECTED_BIT, wifi_event_handler(), WIFI_FAIL_BIT, WIFI_PASSWORD, WIFI_SSID, and WIFI_TIMEOUT_MS.
Referenced by app_main().

5.64.3 Variable Documentation

5.64.3.1 DEFAULT_PIN

```
const uint8_t DEFAULT_PIN[] = "000000000"  [static]
```

Demo PIN — replace with the PIN used during card initialisation.

Examples

BasicUsage/main/main.cpp.

Definition at line 175 of file main.cpp.
Referenced by run_basic_usage_loop().

5.64.3.2 DEFAULT_PIN_LEN

```
const size_t DEFAULT_PIN_LEN = sizeof(DEFAULT_PIN) - 1U  [static]
```

Examples

BasicUsage/main/main.cpp.

Definition at line 176 of file main.cpp.
Referenced by run_basic_usage_loop().

5.64.3.3 LOOP_DELAY_MS

```
const uint32_t LOOP_DELAY_MS = 1000U  [static]
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 73 of file main.cpp.
Referenced by run_basic_usage_loop(), run_connect_loop(), run_sign_loop(), and run_verify_pin_loop().

5.64.3.4 s_retry_num

```
int s_retry_num = 0  [static]
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 81 of file main.cpp.
Referenced by eth_rpc_wifi_connect(), wifi_event_handler(), wifi_event_handler(), and wifi_start().

5.64.3.5 s_wifi_event_group

```
EventGroupHandle_t s_wifi_event_group  [static]
```

Examples

BasicUsage/main/main.cpp, Connect/main/main.cpp, Sign/main/main.cpp, and VerifyPin/main/main.cpp.

Definition at line 80 of file main.cpp.
Referenced by eth_rpc_wifi_connect(), wifi_event_handler(), wifi_event_handler(), and wifi_start().

5.64.3.6 TAG

```
const char* const TAG = "basic_usage"  [static]
```

Definition at line 72 of file main.cpp.

5.64.3.7 WIFI_MAX_RETRY

```
const int WIFI_MAX_RETRY = 5  [static]
```

Definition at line 75 of file main.cpp.

5.64.3.8 WIFI_TIMEOUT_MS

```
const uint32_t WIFI_TIMEOUT_MS = 10000U  [static]
```

Definition at line 74 of file main.cpp.

5.65 main.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00024
00025 #include <string.h>
00026 #include "freertos/FreeRTOS.h"
00027 #include "freertos/task.h"
00028 #include "freertos/event_groups.h"
00029 #include "driver/spi_master.h"
00030 #include "esp_log.h"
00031 #include "esp_wifi.h"
00032 #include "esp_netif.h"
00033 #include "esp_event.h"
00034 #include "nvs_flash.h"
00035 #include "pn532.h"
00036 #include "CryptnoxWallet.h"
00037 #include "Pn532NfcTransport.h"
00038 #include "ESP32Logger.h"
00039 #include "esp32_crypto_provider.h"
00040 #include "ESP32Platform.h"
00041 #include "CW_Utils.h"
00042 #include "config.h"
00043
00044 /* ============================================================================
00045  * 1. Interface Selection
00046  * ----------------------------------------------------------------------------
00047  * Set exactly ONE flag to 1 and the other to 0.
00048  * ============================================================================ */
00049 #define SPI_ENABLED         1
00050 #define I2C_ENABLED         0
00051
00052 /* - SPI wiring -- ESP32-S3 dev kit + Keyestudio PN532 breakout --- */
00053 #if SPI_ENABLED
00054 #define SPI_MOSI            11
00055 #define SPI_MISO            13
00056 #define SPI_SCLK            12
00057 #define SPI_MAX_TRANSFER_SZ 4096
00058 #define SPI_PIN_UNUSED      (-1)
00059 #define NFC_CS              10
00060 #endif
00061
00062 /* - I²C wiring -- Cheap Yellow Display (ESP32) CN1 connector --- */
00063 #if I2C_ENABLED
00064 #define PN532_I2C_PORT      0           /* I2C_NUM_0 */
00065 #define PN532_SDA           27          /* CN1 SDA */
00066 #define PN532_SCL           22          /* CN1 SCL */
00067 #define PN532_IRQ           (-1)        /* unused */
00068 #define PN532_RST           (-1)        /* unused */
00069 #define PN532_I2C_HZ        100000U
00070 #endif
00071
00072 static const char *const TAG           = "basic_usage";
00073 static const uint32_t    LOOP_DELAY_MS  = 1000U;
00074 static const uint32_t    WIFI_TIMEOUT_MS = 10000U;
00075 static const int         WIFI_MAX_RETRY  = 5;
00076
00077 #define WIFI_CONNECTED_BIT BIT0
00078 #define WIFI_FAIL_BIT      BIT1
00079
00080 static EventGroupHandle_t s_wifi_event_group;
00081 static int                s_retry_num = 0;
00082
00095 static void wifi_event_handler(void *arg, esp_event_base_t event_base,
00096                                int32_t event_id, void *event_data)
00097 {
00098     (void)arg;
00099     (void)event_data;
00100     if ((event_base == WIFI_EVENT) && (event_id == WIFI_EVENT_STA_START)) {
00101         esp_wifi_connect();
00102     } else if ((event_base == WIFI_EVENT) &&
00103                (event_id == WIFI_EVENT_STA_DISCONNECTED)) {
00104         if (s_retry_num < WIFI_MAX_RETRY) {
```

```
00105             esp_wifi_connect();
00106             s_retry_num++;
00107         } else {
00108             xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
00109         }
00110     } else if ((event_base == IP_EVENT) && (event_id == IP_EVENT_STA_GOT_IP)) {
00111         s_retry_num = 0;
00112         xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
00113     } else {
00114         /* other events ignored */
00115     }
00116 }
00117
00130 static void wifi_start(void)
00131 {
00132     s_wifi_event_group = xEventGroupCreate();
00133     s_retry_num = 0;
00134     ESP_ERROR_CHECK(esp_netif_init());
00135     ESP_ERROR_CHECK(esp_event_loop_create_default());
00136     (void)esp_netif_create_default_wifi_sta();
00137     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
00138     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
00139     esp_event_handler_instance_t h_any;
00140     esp_event_handler_instance_t h_ip;
00141     ESP_ERROR_CHECK(esp_event_handler_instance_register(
00142         WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler, NULL, &h_any));
00143     ESP_ERROR_CHECK(esp_event_handler_instance_register(
00144         IP_EVENT, IP_EVENT_STA_GOT_IP, &wifi_event_handler, NULL, &h_ip));
00145     wifi_config_t wifi_cfg;
00146     (void)memset(&wifi_cfg, 0, sizeof(wifi_cfg));
00147     (void)strncpy((char *)wifi_cfg.sta.ssid,     WIFI_SSID,
00148                   sizeof(wifi_cfg.sta.ssid)     - 1U);
00149     (void)strncpy((char *)wifi_cfg.sta.password, WIFI_PASSWORD,
00150                   sizeof(wifi_cfg.sta.password) - 1U);
00151     wifi_cfg.sta.threshold.authmode = WIFI_AUTH_WPA2_PSK;
00152     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
00153     ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_cfg));
00154     ESP_ERROR_CHECK(esp_wifi_start());
00155     EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
00156                                            WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
00157                                            pdFALSE, pdFALSE,
00158                                            pdMS_TO_TICKS(WIFI_TIMEOUT_MS));
00159     if ((bits & WIFI_CONNECTED_BIT) != 0U) {
00160         ESP_LOGI(TAG, "WiFi connected");
00161     } else {
00162         ESP_LOGW(TAG, "WiFi connect failed -- TRNG entropy may be reduced");
00163     }
00164     (void)esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP, h_ip);
00165     (void)esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID, h_any);
00166     vEventGroupDelete(s_wifi_event_group);
00167 }
00168
00169 /* ============================================================================
00170  * 2. Default PIN
00171  * ----------------------------------------------------------------------------
00172  * Must match the PIN set on the card (4-9 ASCII digits).
00173  * ============================================================================ */
00175 static const uint8_t DEFAULT_PIN[]    = "000000000";
00176 static const size_t  DEFAULT_PIN_LEN  = sizeof(DEFAULT_PIN) - 1U;
00177
00178 /**************************************************************
00179  * Main loop
00180  **************************************************************/
00181
00192 static void run_basic_usage_loop(CryptnoxWallet &wallet)
00193 {
00194     bool keep_running = true;
00195
00196     while (keep_running) {
00197         /* - Step 1: Connect to card and establish secure channel - */
00198         CW_SecureSession session{};
00199
00200         if (!wallet.connect(session)) {
00201             ESP_LOGW(TAG, "Card not detected -- hold card to reader");
00202             vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
00203             continue;
00204         }
00205
00206         ESP_LOGI(TAG, "Card connected and secure channel established");
00207
00208         /* - Step 2: Sign a test hash ------------------- */
00209         /* NOTE: Card must have a seed loaded before signing works.
00210          * Use the Cryptnox CLI: cryptnox seed generate */
00211         ESP_LOGI(TAG, "Signing test hash...");
00212
00213         uint8_t testHash[CW_HASH_SIZE];
00214         (void)memset(testHash, 0x01, sizeof(testHash));
```

```
00215
00216         /* Build sign request -- PIN included in sign payload for
00217          * authentication.  Alternatively call verifyPin() first
00218          * and use CW_SIGN_PINLESS_K1 here. */
00219         CW_SignRequest signRequest(session,
00220                                    CW_SIGN_CURR_K1,
00221                                    CW_SIGN_SIG_ECDSA_LOW_S,
00222                                    CW_SIGN_WITH_PIN);
00223         signRequest.hash       = testHash;
00224         signRequest.hashLength = static_cast<uint8_t>(CW_HASH_SIZE);
00225         (void)CW_Utils::safe_memcpy(signRequest.pin, sizeof(signRequest.pin),
00226                                     DEFAULT_PIN, DEFAULT_PIN_LEN);
00227
00228         CW_SignResult signResult = wallet.sign(signRequest);
00229
00230         if (signResult.errorCode == CW_OK) {
00231             ESP_LOGI(TAG, "Signature received (64 bytes raw r||s)");
00232
00233             /* Print first 8 bytes of R for a quick visual check */
00234             ESP_LOG_BUFFER_HEX_LEVEL(TAG,
00235                 &signResult.signature[CW_SIG_R_OFFSET], 8U, ESP_LOG_INFO);
00236             ESP_LOGI(TAG, "s:");
00237             ESP_LOG_BUFFER_HEX_LEVEL(TAG,
00238                 &signResult.signature[CW_SIG_S_OFFSET], 8U, ESP_LOG_INFO);
00239
00240             ESP_LOGI(TAG, "Card processed successfully");
00241
00242         } else if (signResult.errorCode == CW_SIGN_PIN_INCORRECT) {
00243             /* CRITICAL: do NOT retry -- each wrong PIN decrements the
00244              * card's retry counter.  Reaching 0 permanently blocks
00245              * the PIN.  Fix DEFAULT_PIN before flashing again. */
00246             ESP_LOGE(TAG, "Wrong PIN -- halting to protect retry counter");
00247             CW_Utils::secure_wipe(testHash,            sizeof(testHash));
00248             CW_Utils::secure_wipe(signResult.signature, sizeof(signResult.signature));
00249             wallet.disconnect(session);
00250             keep_running = false;
00251             continue;
00252
00253         } else {
00254             ESP_LOGE(TAG, "Sign failed: errorCode = 0x%02X",
00255                      static_cast<unsigned int>(signResult.errorCode));
00256         }
00257
00258         /* - Step 3: Securely wipe sensitive buffers --------- */
00259         CW_Utils::secure_wipe(testHash,            sizeof(testHash));
00260         CW_Utils::secure_wipe(signResult.signature, sizeof(signResult.signature));
00261
00262         /* Always disconnect to reset the reader for the next tap. */
00263         wallet.disconnect(session);
00264
00265         vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
00266     }
00267
00268     /* Halt after a fatal error (wrong PIN). */
00269     while (true) {
00270         vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
00271     }
00272 }
00273
00274 /*****************************************************************
00275  * Entry point
00276  *****************************************************************/
00277
00284 extern "C" void app_main(void)
00285 {
00286     esp_err_t nvs_ret = nvs_flash_init();
00287     if ((nvs_ret == ESP_ERR_NVS_NO_FREE_PAGES) ||
00288         (nvs_ret == ESP_ERR_NVS_NEW_VERSION_FOUND)) {
00289         ESP_ERROR_CHECK(nvs_flash_erase());
00290         nvs_ret = nvs_flash_init();
00291     }
00292     ESP_ERROR_CHECK(nvs_ret);
00293     wifi_start();
00294
00295     pn532_t        nfc     = {};
00296     pn532_config_t nfc_cfg = {};
00297
00298 #if SPI_ENABLED
00299     spi_bus_config_t buscfg = {};
00300     buscfg.mosi_io_num     = SPI_MOSI;
00301     buscfg.miso_io_num     = SPI_MISO;
00302     buscfg.sclk_io_num     = SPI_SCLK;
00303     buscfg.quadwp_io_num   = SPI_PIN_UNUSED;
00304     buscfg.quadhd_io_num   = SPI_PIN_UNUSED;
00305     buscfg.max_transfer_sz = SPI_MAX_TRANSFER_SZ;
00306     ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &buscfg, SPI_DMA_CH_AUTO));
00307
```

```
00308     nfc_cfg.transport     = PN532_TRANSPORT_SPI;
00309     nfc_cfg.spi_host      = SPI2_HOST;
00310     nfc_cfg.pin_cs        = NFC_CS;
00311     nfc_cfg.skip_bus_init = true;
00312 #endif
00313
00314 #if I2C_ENABLED
00315     nfc_cfg.transport    = PN532_TRANSPORT_I2C;
00316     nfc_cfg.i2c_port     = PN532_I2C_PORT;
00317     nfc_cfg.pin_sda      = PN532_SDA;
00318     nfc_cfg.pin_scl      = PN532_SCL;
00319     nfc_cfg.pin_irq      = PN532_IRQ;
00320     nfc_cfg.pin_rst      = PN532_RST;
00321     nfc_cfg.i2c_clock_hz = PN532_I2C_HZ;
00322 #endif
00323
00324     esp_err_t nfc_ret = pn532_init(&nfc, &nfc_cfg);
00325     if (nfc_ret != ESP_OK) {
00326         ESP_LOGE(TAG, "PN532 init failed -- check wiring and interface selection");
00327         return;
00328     }
00329
00330     ESP32Logger          logger;
00331     (void)logger.begin(115200UL);
00332
00333     ESP32CryptoProvider  cryptoProvider;
00334     ESP32Platform        platform;
00335     Pn532NfcTransport    nfcTransport(&nfc, logger);
00336     CryptnoxWallet       wallet(nfcTransport, logger, cryptoProvider, platform);
00337
00338     if (!wallet.begin()) {
00339         ESP_LOGE(TAG, "Wallet init failed (SAMConfig)");
00340         return;
00341     }
00342
00343     /* Print PN532 firmware version as a sanity check. */
00344     (void)nfcTransport.printFirmwareVersion();
00345
00346     ESP_LOGI(TAG, "Ready -- hold Cryptnox card to reader");
00347
00348     run_basic_usage_loop(wallet);
00349 }
```

5.66 examples/Connect/main/main.cpp File Reference

```
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "driver/spi_master.h"
#include "esp_log.h"
#include "esp_wifi.h"
#include "esp_netif.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "pn532.h"
#include "CryptnoxWallet.h"
#include "Pn532NfcTransport.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "ESP32Platform.h"
#include "config.h"
```

Include dependency graph for main.cpp:

Macros

- #define SPI_MOSI 11
- #define SPI_MISO 13
- #define SPI_SCLK 12
- #define SPI_MAX_TRANSFER_SZ 4096
- #define SPI_PIN_UNUSED (-1)
- #define NFC_CS 10
- #define WIFI_CONNECTED_BIT BIT0
- #define WIFI_FAIL_BIT BIT1

Functions

- static void wifi_event_handler (void *arg, esp_event_base_t event_base, int32_t event_id, void *event_data)

  FreeRTOS event handler driving the Wi-Fi station state machine.
- static void wifi_start (void)

  Initialise Wi-Fi station mode and block until connected or timeout.
- static void run_connect_loop (CryptnoxWallet &wallet)

  Main application loop: connect, fetch card info, and disconnect.
- void app_main (void)

  ESP-IDF application entry point.

Variables

- static const char *const TAG = "connect"
- static const uint32_t LOOP_DELAY_MS = 1000U
- static const uint32_t WIFI_TIMEOUT_MS = 10000U
- static const int WIFI_MAX_RETRY = 5
- static EventGroupHandle_t s_wifi_event_group
- static int s_retry_num = 0

5.66.1 Macro Definition Documentation

5.66.1.1 NFC_CS

```
#define NFC_CS 10
```

Definition at line 52 of file main.cpp.

5.66.1.2 SPI_MAX_TRANSFER_SZ

```
#define SPI_MAX_TRANSFER_SZ 4096
```

Definition at line 50 of file main.cpp.

5.66.1.3 SPI_MISO

```
#define SPI_MISO 13
```

Definition at line 48 of file main.cpp.

5.66.1.4 SPI_MOSI

```
#define SPI_MOSI 11
```

Definition at line 47 of file main.cpp.

5.66.1.5 SPI_PIN_UNUSED

```
#define SPI_PIN_UNUSED (-1)
```
Definition at line 51 of file main.cpp.

5.66.1.6 SPI_SCLK

```
#define SPI_SCLK 12
```
Definition at line 49 of file main.cpp.

5.66.1.7 WIFI_CONNECTED_BIT

```
#define WIFI_CONNECTED_BIT BIT0
```
Definition at line 59 of file main.cpp.

5.66.1.8 WIFI_FAIL_BIT

```
#define WIFI_FAIL_BIT BIT1
```
Definition at line 60 of file main.cpp.

5.66.2 Function Documentation

5.66.2.1 app_main()

```
void app_main (
            void )
```
ESP-IDF application entry point.
Initialises NVS, starts Wi-Fi for full TRNG entropy, brings up the SPI bus and PN532 reader, then enters run_connect_loop.
Definition at line 192 of file main.cpp.
References ESP32Logger::begin(), NFC_CS, pn532_config_t::pin_cs, pn532_init(), Pn532NfcTransport::printFirmwareV run_connect_loop(), pn532_config_t::skip_bus_init, pn532_config_t::spi_host, SPI_MAX_TRANSFER_SZ, SPI_MISO, SPI_MOSI, SPI_PIN_UNUSED, SPI_SCLK, TAG, and wifi_start().

5.66.2.2 run_connect_loop()

```
void run_connect_loop (
            CryptnoxWallet & wallet)  [static]
```
Main application loop: connect, fetch card info, and disconnect.
Each iteration calls CryptnoxWallet::connect to establish the secure channel, retrieves owner info via CryptnoxWallet::getCardInfo, then tears down the channel with CryptnoxWallet::disconnect.

Parameters

in	*wallet*	Initialised wallet instance.

Examples

Connect/main/main.cpp.

Definition at line 160 of file main.cpp.
References LOOP_DELAY_MS, and TAG.
Referenced by app_main().

5.66.2.3 wifi_event_handler()

```
void wifi_event_handler (
            void * arg,
            esp_event_base_t event_base,
```

```
            int32_t event_id,
            void * event_data)  [static]
```

FreeRTOS event handler driving the Wi-Fi station state machine.
Handles `WIFI_EVENT_STA_START` (triggers connection), `WIFI_EVENT_STA_DISCONNECTED` (retries up to `WIFI_MAX_RETRY` times), and `IP_EVENT_STA_GOT_IP` (signals success via the event group).

Parameters

`in`	*arg*	Unused.
`in`	*event_base*	Event base (`WIFI_EVENT` or `IP_EVENT`).
`in`	*event_id*	Event identifier within `event_base`.
`in`	*event_data*	Event-specific data (unused).

Definition at line 77 of file main.cpp.
References s_retry_num, s_wifi_event_group, WIFI_CONNECTED_BIT, WIFI_FAIL_BIT, and WIFI_MAX_RETRY.

5.66.2.4 wifi_start()

```
void wifi_start (
            void )  [static]
```

Initialise Wi-Fi station mode and block until connected or timeout.
Starts the ESP Wi-Fi stack, registers wifi_event_handler, configures the SSID and password from WIFI_SSID / WIFI_PASSWORD in `config.h`, then waits up to `WIFI_TIMEOUT_MS` for an IP address. The radio must be active before crypto operations so the hardware TRNG operates with full entropy (SEC-001).

Note

If the connection fails the function logs a warning and returns normally; the firmware continues with reduced TRNG entropy.

Definition at line 112 of file main.cpp.
References s_retry_num, s_wifi_event_group, TAG, WIFI_CONNECTED_BIT, wifi_event_handler(), WIFI_FAIL_BIT, WIFI_PASSWORD, WIFI_SSID, and WIFI_TIMEOUT_MS.

5.66.3 Variable Documentation

5.66.3.1 LOOP_DELAY_MS

```
const uint32_t LOOP_DELAY_MS = 1000U  [static]
```

Definition at line 55 of file main.cpp.

5.66.3.2 s_retry_num

```
int s_retry_num = 0  [static]
```

Definition at line 63 of file main.cpp.

5.66.3.3 s_wifi_event_group

```
EventGroupHandle_t s_wifi_event_group  [static]
```

Definition at line 62 of file main.cpp.

5.66.3.4 TAG

```
const char* const TAG = "connect"  [static]
```

Definition at line 54 of file main.cpp.

5.66.3.5 WIFI_MAX_RETRY

```
const int WIFI_MAX_RETRY = 5  [static]
```
Definition at line 57 of file main.cpp.

5.66.3.6 WIFI_TIMEOUT_MS

```
const uint32_t WIFI_TIMEOUT_MS = 10000U  [static]
```
Definition at line 56 of file main.cpp.

5.67 main.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00027
00028 #include <string.h>
00029 #include "freertos/FreeRTOS.h"
00030 #include "freertos/task.h"
00031 #include "freertos/event_groups.h"
00032 #include "driver/spi_master.h"
00033 #include "esp_log.h"
00034 #include "esp_wifi.h"
00035 #include "esp_netif.h"
00036 #include "esp_event.h"
00037 #include "nvs_flash.h"
00038 #include "pn532.h"
00039 #include "CryptnoxWallet.h"
00040 #include "Pn532NfcTransport.h"
00041 #include "ESP32Logger.h"
00042 #include "esp32_crypto_provider.h"
00043 #include "ESP32Platform.h"
00044 #include "config.h"
00045
00046 /* - SPI wiring -- ESP32-S3 dev kit + Keyestudio PN532 breakout --- */
00047 #define SPI_MOSI            11
00048 #define SPI_MISO            13
00049 #define SPI_SCLK            12
00050 #define SPI_MAX_TRANSFER_SZ 4096
00051 #define SPI_PIN_UNUSED      (-1)
00052 #define NFC_CS              10
00053
00054 static const char *const TAG           = "connect";
00055 static const uint32_t    LOOP_DELAY_MS  = 1000U;
00056 static const uint32_t    WIFI_TIMEOUT_MS = 10000U;
00057 static const int         WIFI_MAX_RETRY  = 5;
00058
00059 #define WIFI_CONNECTED_BIT BIT0
00060 #define WIFI_FAIL_BIT      BIT1
00061
00062 static EventGroupHandle_t s_wifi_event_group;
00063 static int                s_retry_num = 0;
00064
00077 static void wifi_event_handler(void *arg, esp_event_base_t event_base,
00078                                int32_t event_id, void *event_data)
00079 {
00080     (void)arg;
00081     (void)event_data;
00082     if ((event_base == WIFI_EVENT) && (event_id == WIFI_EVENT_STA_START)) {
00083         esp_wifi_connect();
00084     } else if ((event_base == WIFI_EVENT) &&
00085                (event_id == WIFI_EVENT_STA_DISCONNECTED)) {
00086         if (s_retry_num < WIFI_MAX_RETRY) {
00087             esp_wifi_connect();
00088             s_retry_num++;
00089         } else {
00090             xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
00091         }
00092     } else if ((event_base == IP_EVENT) && (event_id == IP_EVENT_STA_GOT_IP)) {
00093         s_retry_num = 0;
00094         xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
00095     } else {
00096         /* other events ignored */
00097     }
00098 }
00099
00112 static void wifi_start(void)
```

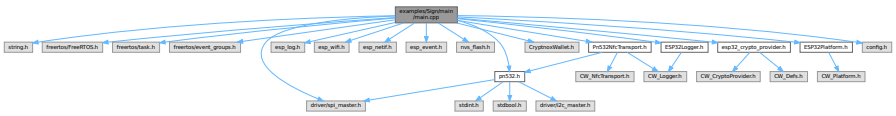
```
00113 {
00114     s_wifi_event_group = xEventGroupCreate();
00115     s_retry_num = 0;
00116     ESP_ERROR_CHECK(esp_netif_init());
00117     ESP_ERROR_CHECK(esp_event_loop_create_default());
00118     (void)esp_netif_create_default_wifi_sta();
00119     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
00120     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
00121     esp_event_handler_instance_t h_any;
00122     esp_event_handler_instance_t h_ip;
00123     ESP_ERROR_CHECK(esp_event_handler_instance_register(
00124         WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler, NULL, &h_any));
00125     ESP_ERROR_CHECK(esp_event_handler_instance_register(
00126         IP_EVENT, IP_EVENT_STA_GOT_IP, &wifi_event_handler, NULL, &h_ip));
00127     wifi_config_t wifi_cfg;
00128     (void)memset(&wifi_cfg, 0, sizeof(wifi_cfg));
00129     (void)strncpy((char *)wifi_cfg.sta.ssid,     WIFI_SSID,
00130                   sizeof(wifi_cfg.sta.ssid)     - 1U);
00131     (void)strncpy((char *)wifi_cfg.sta.password, WIFI_PASSWORD,
00132                   sizeof(wifi_cfg.sta.password) - 1U);
00133     wifi_cfg.sta.threshold.authmode = WIFI_AUTH_WPA2_PSK;
00134     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
00135     ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_cfg));
00136     ESP_ERROR_CHECK(esp_wifi_start());
00137     EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
00138                                            WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
00139                                            pdFALSE, pdFALSE,
00140                                            pdMS_TO_TICKS(WIFI_TIMEOUT_MS));
00141     if ((bits & WIFI_CONNECTED_BIT) != 0U) {
00142         ESP_LOGI(TAG, "WiFi connected");
00143     } else {
00144         ESP_LOGW(TAG, "WiFi connect failed -- TRNG entropy may be reduced");
00145     }
00146     (void)esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP, h_ip);
00147     (void)esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID, h_any);
00148     vEventGroupDelete(s_wifi_event_group);
00149 }
00150
00160 static void run_connect_loop(CryptnoxWallet &wallet)
00161 {
00162     while (true) {
00163         CW_SecureSession session{};
00164         bool connected = wallet.connect(session);
00165
00166         if (connected) {
00167             ESP_LOGI(TAG, "Card connected, secure channel established");
00168
00169             CW_CardInfo info{};
00170             bool infoOk = wallet.getCardInfo(session, &info);
00171             if (infoOk) {
00172                 ESP_LOGI(TAG, "Owner name : %s", info.name);
00173                 ESP_LOGI(TAG, "Owner email: %s", info.email);
00174             } else {
00175                 ESP_LOGW(TAG, "getCardInfo failed");
00176             }
00177         } else {
00178             ESP_LOGW(TAG, "Card not detected or secure channel failed");
00179         }
00180
00181         wallet.disconnect(session);
00182         vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
00183     }
00184 }
00185
00192 extern "C" void app_main(void)
00193 {
00194     esp_err_t nvs_ret = nvs_flash_init();
00195     if ((nvs_ret == ESP_ERR_NVS_NO_FREE_PAGES) ||
00196         (nvs_ret == ESP_ERR_NVS_NEW_VERSION_FOUND)) {
00197         ESP_ERROR_CHECK(nvs_flash_erase());
00198         nvs_ret = nvs_flash_init();
00199     }
00200     ESP_ERROR_CHECK(nvs_ret);
00201     wifi_start();
00202
00203     spi_bus_config_t buscfg = {};
00204     buscfg.mosi_io_num     = SPI_MOSI;
00205     buscfg.miso_io_num     = SPI_MISO;
00206     buscfg.sclk_io_num     = SPI_SCLK;
00207     buscfg.quadwp_io_num   = SPI_PIN_UNUSED;
00208     buscfg.quadhd_io_num   = SPI_PIN_UNUSED;
00209     buscfg.max_transfer_sz = SPI_MAX_TRANSFER_SZ;
00210     ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &buscfg, SPI_DMA_CH_AUTO));
00211
00212     pn532_t nfc = {};
00213     pn532_config_t nfc_cfg = {};
00214     nfc_cfg.spi_host      = SPI2_HOST;
```

```
00215     nfc_cfg.pin_cs        = NFC_CS;
00216     nfc_cfg.skip_bus_init = true;
00217     esp_err_t nfc_ret = pn532_init(&nfc, &nfc_cfg);
00218
00219     if (nfc_ret == ESP_OK) {
00220         ESP32Logger logger;
00221         (void)logger.begin(115200UL);
00222
00223         ESP32CryptoProvider cryptoProvider;
00224         ESP32Platform platform;
00225         Pn532NfcTransport nfcTransport(&nfc, logger);
00226         CryptnoxWallet wallet(nfcTransport, logger, cryptoProvider, platform);
00227
00228         if (wallet.begin()) {
00229             (void)nfcTransport.printFirmwareVersion();
00230             run_connect_loop(wallet);
00231         } else {
00232             ESP_LOGE(TAG, "Wallet init failed");
00233         }
00234     } else {
00235         ESP_LOGE(TAG, "PN532 init failed");
00236     }
00237 }
```

5.68 examples/Sign/main/main.cpp File Reference

```
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "driver/spi_master.h"
#include "esp_log.h"
#include "esp_wifi.h"
#include "esp_netif.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "pn532.h"
#include "CryptnoxWallet.h"
#include "Pn532NfcTransport.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "ESP32Platform.h"
#include "config.h"
```

Include dependency graph for main.cpp:



Macros

- #define SPI_MOSI 11
- #define SPI_MISO 13
- #define SPI_SCLK 12
- #define SPI_MAX_TRANSFER_SZ 4096
- #define SPI_PIN_UNUSED (-1)
- #define NFC_CS 10
- #define WIFI_CONNECTED_BIT BIT0
- #define WIFI_FAIL_BIT BIT1

Functions

- static void wifi_event_handler (void *arg, esp_event_base_t event_base, int32_t event_id, void *event_data)

  FreeRTOS event handler driving the Wi-Fi station state machine.
- static void wifi_start (void)

  Initialise Wi-Fi station mode and block until connected or timeout.
- static void run_sign_loop (CryptnoxWallet &wallet)

  Main application loop: connect, sign a test hash, and disconnect.
- void app_main (void)

  ESP-IDF application entry point.

Variables

- static const char *const TAG = "sign"
- static const uint32_t LOOP_DELAY_MS = 1000U
- static const uint32_t WIFI_TIMEOUT_MS = 10000U
- static const int WIFI_MAX_RETRY = 5
- static EventGroupHandle_t s_wifi_event_group
- static int s_retry_num = 0

5.68.1 Macro Definition Documentation

5.68.1.1 NFC_CS

```
#define NFC_CS 10
```

Definition at line 60 of file main.cpp.

5.68.1.2 SPI_MAX_TRANSFER_SZ

```
#define SPI_MAX_TRANSFER_SZ 4096
```

Definition at line 58 of file main.cpp.

5.68.1.3 SPI_MISO

```
#define SPI_MISO 13
```

Definition at line 56 of file main.cpp.

5.68.1.4 SPI_MOSI

```
#define SPI_MOSI 11
```

Definition at line 55 of file main.cpp.

5.68.1.5 SPI_PIN_UNUSED

```
#define SPI_PIN_UNUSED (-1)
```

Definition at line 59 of file main.cpp.

5.68.1.6 SPI_SCLK

```
#define SPI_SCLK 12
```

Definition at line 57 of file main.cpp.

5.68.1.7 WIFI_CONNECTED_BIT

```
#define WIFI_CONNECTED_BIT BIT0
```

Definition at line 67 of file main.cpp.

5.68.1.8 WIFI_FAIL_BIT

```
#define WIFI_FAIL_BIT BIT1
```

Definition at line 68 of file main.cpp.

5.68.2 Function Documentation

5.68.2.1 app_main()

```
void app_main (
            void )
```

ESP-IDF application entry point.
Initialises NVS, starts Wi-Fi for full TRNG entropy, brings up the SPI bus and PN532 reader, then enters run_sign_loop.
Definition at line 241 of file main.cpp.
References ESP32Logger::begin(), NFC_CS, pn532_config_t::pin_cs, pn532_init(), run_sign_loop(), pn532_config_t::skip_bus_init, pn532_config_t::spi_host, SPI_MAX_TRANSFER_SZ, SPI_MISO, SPI_MOSI, SPI_PIN_UNUSED, SPI_SCLK, TAG, and wifi_start().

5.68.2.2 run_sign_loop()

```
void run_sign_loop (
            CryptnoxWallet & wallet)  [static]
```

Main application loop: connect, sign a test hash, and disconnect.
Each iteration establishes the secure channel, signs a 32-byte test hash on the secp256k1 curve via CryptnoxWallet::sign, then disconnects. Halts permanently on CW_SIGN_PIN_INCORRECT to protect the card's retry counter.

Parameters

in	*wallet*	Initialised wallet instance.

Examples

Sign/main/main.cpp.

Definition at line 169 of file main.cpp.
References LOOP_DELAY_MS, and TAG.
Referenced by app_main().

5.68.2.3 wifi_event_handler()

```
void wifi_event_handler (
            void * arg,
            esp_event_base_t event_base,
            int32_t event_id,
            void * event_data)  [static]
```

FreeRTOS event handler driving the Wi-Fi station state machine.
Handles `WIFI_EVENT_STA_START` (triggers connection), `WIFI_EVENT_STA_DISCONNECTED` (retries up to `WIFI_MAX_RETRY` times), and `IP_EVENT_STA_GOT_IP` (signals success via the event group).

Parameters

in	*arg*	Unused.
in	*event_base*	Event base (`WIFI_EVENT` or `IP_EVENT`).
in	*event_id*	Event identifier within `event_base`.

in	*event_data*	Event-specific data (unused).

Definition at line 85 of file main.cpp.
References s_retry_num, s_wifi_event_group, WIFI_CONNECTED_BIT, WIFI_FAIL_BIT, and WIFI_MAX_RETRY.

5.68.2.4 wifi_start()

```
void wifi_start (
            void )  [static]
```

Initialise Wi-Fi station mode and block until connected or timeout.
Starts the ESP Wi-Fi stack, registers wifi_event_handler, configures the SSID and password from WIFI_SSID / WIFI_PASSWORD in `config.h`, then waits up to `WIFI_TIMEOUT_MS` for an IP address. The radio must be active before crypto operations so the hardware TRNG operates with full entropy (SEC-001).

Note

If the connection fails the function logs a warning and returns normally; the firmware continues with reduced TRNG entropy.

Definition at line 120 of file main.cpp.
References s_retry_num, s_wifi_event_group, TAG, WIFI_CONNECTED_BIT, wifi_event_handler(), WIFI_FAIL_BIT, WIFI_PASSWORD, WIFI_SSID, and WIFI_TIMEOUT_MS.

5.68.3 Variable Documentation

5.68.3.1 LOOP_DELAY_MS

```
const uint32_t LOOP_DELAY_MS = 1000U  [static]
```

Definition at line 63 of file main.cpp.

5.68.3.2 s_retry_num

```
int s_retry_num = 0  [static]
```

Definition at line 71 of file main.cpp.

5.68.3.3 s_wifi_event_group

```
EventGroupHandle_t s_wifi_event_group  [static]
```

Definition at line 70 of file main.cpp.

5.68.3.4 TAG

```
const char* const TAG = "sign"  [static]
```

Definition at line 62 of file main.cpp.

5.68.3.5 WIFI_MAX_RETRY

```
const int WIFI_MAX_RETRY = 5  [static]
```

Definition at line 65 of file main.cpp.

5.68.3.6 WIFI_TIMEOUT_MS

```
const uint32_t WIFI_TIMEOUT_MS = 10000U  [static]
```

Definition at line 64 of file main.cpp.

5.69 main.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00035
00036 #include <string.h>
00037 #include "freertos/FreeRTOS.h"
00038 #include "freertos/task.h"
00039 #include "freertos/event_groups.h"
00040 #include "driver/spi_master.h"
00041 #include "esp_log.h"
00042 #include "esp_wifi.h"
00043 #include "esp_netif.h"
00044 #include "esp_event.h"
00045 #include "nvs_flash.h"
00046 #include "pn532.h"
00047 #include "CryptnoxWallet.h"
00048 #include "Pn532NfcTransport.h"
00049 #include "ESP32Logger.h"
00050 #include "esp32_crypto_provider.h"
00051 #include "ESP32Platform.h"
00052 #include "config.h"
00053
00054 /* - SPI wiring -- ESP32-S3 dev kit + Keyestudio PN532 breakout --- */
00055 #define SPI_MOSI            11
00056 #define SPI_MISO            13
00057 #define SPI_SCLK            12
00058 #define SPI_MAX_TRANSFER_SZ 4096
00059 #define SPI_PIN_UNUSED      (-1)
00060 #define NFC_CS              10
00061
00062 static const char *const TAG           = "sign";
00063 static const uint32_t    LOOP_DELAY_MS  = 1000U;
00064 static const uint32_t    WIFI_TIMEOUT_MS = 10000U;
00065 static const int         WIFI_MAX_RETRY  = 5;
00066
00067 #define WIFI_CONNECTED_BIT BIT0
00068 #define WIFI_FAIL_BIT      BIT1
00069
00070 static EventGroupHandle_t s_wifi_event_group;
00071 static int                s_retry_num = 0;
00072
00085 static void wifi_event_handler(void *arg, esp_event_base_t event_base,
00086                                int32_t event_id, void *event_data)
00087 {
00088     (void)arg;
00089     (void)event_data;
00090     if ((event_base == WIFI_EVENT) && (event_id == WIFI_EVENT_STA_START)) {
00091         esp_wifi_connect();
00092     } else if ((event_base == WIFI_EVENT) &&
00093                (event_id == WIFI_EVENT_STA_DISCONNECTED)) {
00094         if (s_retry_num < WIFI_MAX_RETRY) {
00095             esp_wifi_connect();
00096             s_retry_num++;
00097         } else {
00098             xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
00099         }
00100     } else if ((event_base == IP_EVENT) && (event_id == IP_EVENT_STA_GOT_IP)) {
00101         s_retry_num = 0;
00102         xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
00103     } else {
00104         /* other events ignored */
00105     }
00106 }
00107
00120 static void wifi_start(void)
00121 {
00122     s_wifi_event_group = xEventGroupCreate();
00123     s_retry_num = 0;
00124     ESP_ERROR_CHECK(esp_netif_init());
00125     ESP_ERROR_CHECK(esp_event_loop_create_default());
00126     (void)esp_netif_create_default_wifi_sta();
00127     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
00128     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
00129     esp_event_handler_instance_t h_any;
00130     esp_event_handler_instance_t h_ip;
00131     ESP_ERROR_CHECK(esp_event_handler_instance_register(
00132         WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler, NULL, &h_any));
00133     ESP_ERROR_CHECK(esp_event_handler_instance_register(
00134         IP_EVENT, IP_EVENT_STA_GOT_IP, &wifi_event_handler, NULL, &h_ip));
00135     wifi_config_t wifi_cfg;
00136     (void)memset(&wifi_cfg, 0, sizeof(wifi_cfg));
```

```
00137     (void)strncpy((char *)wifi_cfg.sta.ssid,     WIFI_SSID,
00138                   sizeof(wifi_cfg.sta.ssid)     - 1U);
00139     (void)strncpy((char *)wifi_cfg.sta.password, WIFI_PASSWORD,
00140                   sizeof(wifi_cfg.sta.password) - 1U);
00141     wifi_cfg.sta.threshold.authmode = WIFI_AUTH_WPA2_PSK;
00142     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
00143     ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_cfg));
00144     ESP_ERROR_CHECK(esp_wifi_start());
00145     EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
00146                                            WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
00147                                            pdFALSE, pdFALSE,
00148                                            pdMS_TO_TICKS(WIFI_TIMEOUT_MS));
00149     if ((bits & WIFI_CONNECTED_BIT) != 0U) {
00150         ESP_LOGI(TAG, "WiFi connected");
00151     } else {
00152         ESP_LOGW(TAG, "WiFi connect failed -- TRNG entropy may be reduced");
00153     }
00154     (void)esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP, h_ip);
00155     (void)esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID, h_any);
00156     vEventGroupDelete(s_wifi_event_group);
00157 }
00158
00169 static void run_sign_loop(CryptnoxWallet &wallet)
00170 {
00171     /* Replace with the SHA-256 (or Keccak-256) digest of the real transaction. */
00172     static const uint8_t TEST_HASH[CW_HASH_SIZE] = {
00173         0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U,
00174         0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U,
00175         0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U,
00176         0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U,
00177     };
00178     /* Replace with the PIN set on the card (4-9 ASCII digits). */
00179     static const uint8_t DEMO_PIN[CW_MAX_PIN_LENGTH] = {
00180         '0', '0', '0', '0', '0', '0', '0', '0', '0'
00181     };
00182
00183     bool keep_running = true;
00184     while (keep_running) {
00185         CW_SecureSession session{};
00186         bool connected = wallet.connect(session);
00187
00188         if (!connected) {
00189             ESP_LOGW(TAG, "Card not detected");
00190         }
00191
00192         if (connected) {
00193             CW_SignRequest req(session,
00194                                CW_SIGN_CURR_K1,
00195                                CW_SIGN_SIG_ECDSA_LOW_S,
00196                                CW_SIGN_WITH_PIN);
00197             req.hash       = TEST_HASH;
00198             req.hashLength = static_cast<uint8_t>(CW_HASH_SIZE);
00199             (void)CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
00200                                         DEMO_PIN, CW_MAX_PIN_LENGTH);
00201
00202             CW_SignResult result = wallet.sign(req);
00203
00204             if (result.errorCode == CW_OK) {
00205                 ESP_LOGI(TAG, "Sign OK - r:");
00206                 ESP_LOG_BUFFER_HEX_LEVEL(TAG, &result.signature[CW_SIG_R_OFFSET],
00207                                          CW_HASH_SIZE, ESP_LOG_INFO);
00208                 ESP_LOGI(TAG, "s:");
00209                 ESP_LOG_BUFFER_HEX_LEVEL(TAG, &result.signature[CW_SIG_S_OFFSET],
00210                                          CW_HASH_SIZE, ESP_LOG_INFO);
00211             } else if (result.errorCode == CW_SIGN_PIN_INCORRECT) {
00212                 /* CRITICAL: do NOT retry -- each wrong PIN decrements the card's
00213                  * retry counter. Reaching 0 permanently blocks the PIN.
00214                  * Fix DEMO_PIN before flashing again. */
00215                 ESP_LOGE(TAG, "Wrong PIN -- halting to protect retry counter");
00216                 keep_running = false;
00217             } else {
00218                 ESP_LOGE(TAG, "Sign failed: 0x%02X",
00219                          static_cast<unsigned int>(result.errorCode));
00220             }
00221         }
00222
00223         wallet.disconnect(session);
00224
00225         if (keep_running) {
00226             vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
00227         }
00228     }
00229
00230     while (true) {
00231         vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
00232     }
00233 }
```

```
00234
00241 extern "C" void app_main(void)
00242 {
00243     esp_err_t nvs_ret = nvs_flash_init();
00244     if ((nvs_ret == ESP_ERR_NVS_NO_FREE_PAGES) ||
00245         (nvs_ret == ESP_ERR_NVS_NEW_VERSION_FOUND)) {
00246         ESP_ERROR_CHECK(nvs_flash_erase());
00247         nvs_ret = nvs_flash_init();
00248     }
00249     ESP_ERROR_CHECK(nvs_ret);
00250     wifi_start();
00251
00252     spi_bus_config_t buscfg = {};
00253     buscfg.mosi_io_num     = SPI_MOSI;
00254     buscfg.miso_io_num     = SPI_MISO;
00255     buscfg.sclk_io_num     = SPI_SCLK;
00256     buscfg.quadwp_io_num   = SPI_PIN_UNUSED;
00257     buscfg.quadhd_io_num   = SPI_PIN_UNUSED;
00258     buscfg.max_transfer_sz = SPI_MAX_TRANSFER_SZ;
00259     ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &buscfg, SPI_DMA_CH_AUTO));
00260
00261     pn532_t nfc = {};
00262     pn532_config_t nfc_cfg = {};
00263     nfc_cfg.spi_host      = SPI2_HOST;
00264     nfc_cfg.pin_cs        = NFC_CS;
00265     nfc_cfg.skip_bus_init = true;
00266     esp_err_t nfc_ret = pn532_init(&nfc, &nfc_cfg);
00267
00268     if (nfc_ret == ESP_OK) {
00269         ESP32Logger logger;
00270         (void)logger.begin(115200UL);
00271
00272         ESP32CryptoProvider cryptoProvider;
00273         ESP32Platform platform;
00274         Pn532NfcTransport nfcTransport(&nfc, logger);
00275         CryptnoxWallet wallet(nfcTransport, logger, cryptoProvider, platform);
00276
00277         if (wallet.begin()) {
00278             run_sign_loop(wallet);
00279         } else {
00280             ESP_LOGE(TAG, "Wallet init failed");
00281         }
00282     } else {
00283         ESP_LOGE(TAG, "PN532 init failed");
00284     }
00285 }
```

5.70 examples/UsdcSigning/main/main.cpp File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <inttypes.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/spi_master.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "CryptnoxWallet.h"
#include "CW_Utils.h"
#include "Pn532NfcTransport.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "ESP32Platform.h"
#include "pn532.h"
#include "keccak256.h"
#include "eth_rlp.h"
#include "eth_rpc.h"
#include "config.h"
```

Include dependency graph for main.cpp:

Macros

- #define SPI_ENABLED 1
- #define I2C_ENABLED 0
- #define SPI_MOSI 11
- #define SPI_MISO 13
- #define SPI_SCLK 12
- #define SPI_MAX_TRANSFER_SZ 4096
- #define SPI_PIN_UNUSED (-1)
- #define NFC_CS 10
- #define TX_BUF_SIZE 300U

Functions

- static void parse_address (const char *hex, uint8_t out[20])

  Parse a hex string into a 20-byte Ethereum address.
- static void build_usdc_calldata (uint8_t out[68], const char *to_hex, uint64_t amount)

  Build the 68-byte ABI-encoded calldata for a USDC `transfer` call.
- static void signing_loop (CryptnoxWallet &wallet)

  Main application loop: sign and broadcast a USDC transfer each card tap.
- void app_main (void)

  ESP-IDF application entry point.

Variables

- static const char *const TAG = "usdc_signing"
- static const uint8_t TRANSFER_SELECTOR [4] = { 0xa9U, 0x05U, 0x9cU, 0xbbU }

5.70.1 Macro Definition Documentation

5.70.1.1 I2C_ENABLED

```
#define I2C_ENABLED 0
```

Definition at line 66 of file main.cpp.

5.70.1.2 NFC_CS

```
#define NFC_CS 10
```

Definition at line 75 of file main.cpp.

5.70.1.3 SPI_ENABLED

```
#define SPI_ENABLED 1
```

Definition at line 65 of file main.cpp.

5.70.1.4 SPI_MAX_TRANSFER_SZ

```
#define SPI_MAX_TRANSFER_SZ 4096
```
Definition at line 73 of file main.cpp.

5.70.1.5 SPI_MISO

```
#define SPI_MISO 13
```
Definition at line 71 of file main.cpp.

5.70.1.6 SPI_MOSI

```
#define SPI_MOSI 11
```
Definition at line 70 of file main.cpp.

5.70.1.7 SPI_PIN_UNUSED

```
#define SPI_PIN_UNUSED (-1)
```
Definition at line 74 of file main.cpp.

5.70.1.8 SPI_SCLK

```
#define SPI_SCLK 12
```
Definition at line 72 of file main.cpp.

5.70.1.9 TX_BUF_SIZE

```
#define TX_BUF_SIZE 300U
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 92 of file main.cpp.
Referenced by signing_loop().

5.70.2 Function Documentation

5.70.2.1 app_main()

```
void app_main (
            void )
```
ESP-IDF application entry point.
Initialises NVS, brings up the PN532 reader (I²C or SPI), connects to Wi-Fi and the Ethereum RPC endpoint, then enters signing_loop.
Definition at line 319 of file main.cpp.
References ADDR_FROM, ESP32Logger::begin(), eth_rpc_init(), eth_rpc_set_auth(), eth_rpc_wifi_connect(), pn532_config_t::i2c_clock_hz, pn532_config_t::i2c_port, NFC_CS, pn532_config_t::pin_cs, pn532_config_t::pin_irq, pn532_config_t::pin_rst, pn532_config_t::pin_scl, pn532_config_t::pin_sda, pn532_init(), PN532_TRANSPORT_I2C, PN532_TRANSPORT_SPI, RPC_URL, signing_loop(), pn532_config_t::skip_bus_init, pn532_config_t::spi_host, SPI_MAX_TRANSFER_SZ, SPI_MISO, SPI_MOSI, SPI_PIN_UNUSED, SPI_SCLK, TAG, pn532_config_t::transport, WIFI_PASSWORD, and WIFI_SSID.

5.70.2.2 build_usdc_calldata()

```
void build_usdc_calldata (
            uint8_t out[68],
            const char * to_hex,
            uint64_t amount)  [static]
```

Build the 68-byte ABI-encoded calldata for a USDC `transfer` call.
Encodes the ERC-20 `transfer(address,uint256)` selector followed by the ABI-encoded arguments:

```
selector(4) | zeroes(12) | to(20) | zeroes(24) | amount_be(8)
```

Parameters

`out`	*out*	68-byte output buffer for the calldata.
`in`	*to_hex*	Recipient address as a hex string (with or without `0x`).
`in`	*amount*	Transfer amount in USDC base units (6 decimals).

Examples

UsdcSigning/main/main.cpp.

Definition at line 140 of file main.cpp.
References parse_address(), and TRANSFER_SELECTOR.
Referenced by signing_loop().

5.70.2.3 parse_address()

```
void parse_address (
            const char * hex,
            uint8_t out[20])  [static]
```

Parse a hex string into a 20-byte Ethereum address.
Accepts an optional `0x` or `0X` prefix. If the source is shorter than 20 bytes the remaining output bytes are zeroed; if longer, the excess is silently discarded.

Parameters

`in`	*hex*	Hex string (with or without `0x` prefix).
`out`	*out*	20-byte output buffer for the decoded address.

Examples

UsdcSigning/main/main.cpp.

Definition at line 108 of file main.cpp.
Referenced by build_usdc_calldata(), and signing_loop().

5.70.2.4 signing_loop()

```
void signing_loop (
            CryptnoxWallet & wallet)  [static]
```

Main application loop: sign and broadcast a USDC transfer each card tap.
Each iteration waits for a card, fetches the current nonce, builds and hashes an EIP-1559 USDC transfer transaction, signs it on the card via CryptnoxWallet::sign, recovers the `v` parity, and broadcasts the signed transaction via `eth_rpc_send_raw_tx`.

Parameters

`in`	*wallet*	Initialised wallet instance.

Examples

UsdcSigning/main/main.cpp.

Definition at line 173 of file main.cpp.
References ADDR_TO, ADDR_USDC, AMOUNT_USDC, build_usdc_calldata(), eth_tx_t::calldata, eth_tx_t::calldata_len, CARD_PIN, CARD_PIN_LEN, eth_tx_t::chain_id, CHAIN_ID_SEPOLIA, eth_rlp_encode_signed(), eth_rlp_encode_unsigned(), eth_rpc_ecrecover_parity(), eth_rpc_get_nonce(), eth_rpc_send_raw_tx(), eth_tx_t::eth_value, eth_tx_t::gas_limit, GAS_LIMIT_ERC20, keccak256(), MAX_FEE, eth_tx_t::max_fee, MAX_PRIORITY_FEE, eth_tx_t::max_priority_fee, eth_tx_t::nonce, parse_address(), TAG, eth_tx_t::to, and TX_BUF_SIZE.
Referenced by app_main().

5.70.3 Variable Documentation

5.70.3.1 TAG

```
const char* const TAG = "usdc_signing"  [static]
```
Definition at line 61 of file main.cpp.

5.70.3.2 TRANSFER_SELECTOR

```
const uint8_t TRANSFER_SELECTOR[4] = { 0xa9U, 0x05U, 0x9cU, 0xbbU }  [static]
```

Examples

UsdcSigning/main/main.cpp.

Definition at line 89 of file main.cpp.
Referenced by build_usdc_calldata().

5.71 main.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00032
00033 #include <stdio.h>
00034 #include <string.h>
00035 #include <stdlib.h>
00036 #include <inttypes.h>
00037
00038 #include "freertos/FreeRTOS.h"
00039 #include "freertos/task.h"
00040 #include "driver/spi_master.h"
00041 #include "driver/gpio.h"
00042 #include "esp_log.h"
00043 #include "nvs_flash.h"
00044
00045 #include "CryptnoxWallet.h"
00046 #include "CW_Utils.h"
00047 #include "Pn532NfcTransport.h"
00048 #include "ESP32Logger.h"
00049 #include "esp32_crypto_provider.h"
00050 #include "ESP32Platform.h"
00051
00052 extern "C" {
00053 #include "pn532.h"
00054 #include "keccak256.h"
00055 #include "eth_rlp.h"
00056 #include "eth_rpc.h"
00057 }
00058
00059 #include "config.h"
00060
00061 static const char *const TAG = "usdc_signing";
00062
00063 /* - PN532 transport selector -------------------------
00064  * Enable exactly one transport by setting its flag to 1 (the other to 0). */
00065 #define SPI_ENABLED         1
00066 #define I2C_ENABLED         0
00067
00068 /* - SPI wiring -- ESP32-S3 dev kit + Keyestudio PN532 breakout --- */
```

```
00069 #if SPI_ENABLED
00070 #define SPI_MOSI            11
00071 #define SPI_MISO            13
00072 #define SPI_SCLK            12
00073 #define SPI_MAX_TRANSFER_SZ 4096
00074 #define SPI_PIN_UNUSED      (-1)
00075 #define NFC_CS              10
00076 #endif
00077
00078 /* - I²C wiring -- Cheap Yellow Display (ESP32) CN1 connector --- */
00079 #if I2C_ENABLED
00080 #define PN532_I2C_PORT      0          /* I2C_NUM_0 */
00081 #define PN532_SDA           27         /* CN1 SDA */
00082 #define PN532_SCL           22         /* CN1 SCL */
00083 #define PN532_IRQ           (-1)       /* unused */
00084 #define PN532_RST           (-1)       /* unused */
00085 #define PN532_I2C_HZ        100000U
00086 #endif
00087
00088 /* - USDC ERC-20 transfer(address,uint256) selector -------- */
00089 static const uint8_t TRANSFER_SELECTOR[4] = { 0xa9U, 0x05U, 0x9cU, 0xbbU };
00090
00091 /* - Unsigned and signed tx buffers (EIP-1559 type 2) ------- */
00092 #define TX_BUF_SIZE 300U
00093
00094 /**********************************************************************
00095  * Helpers
00096  **********************************************************************/
00097
00108 static void parse_address(const char *hex, uint8_t out[20])
00109 {
00110     const char *p = hex;
00111     if ((p[0] == '0') && ((p[1] == 'x') || (p[1] == 'X'))) {
00112         p += 2;
00113     }
00114     (void)memset(out, 0, 20U);
00115     size_t i;
00116     for (i = 0U; (i < 20U) && (p[0] != '\0') && (p[1] != '\0'); i++) {
00117         uint8_t hi = (uint8_t)((*p >= 'a') ? (*p - 'a' + 10) :
00118                                (*p >= 'A') ? (*p - 'A' + 10) : (*p - '0'));
00119         p++;
00120         uint8_t lo = (uint8_t)((*p >= 'a') ? (*p - 'a' + 10) :
00121                                (*p >= 'A') ? (*p - 'A' + 10) : (*p - '0'));
00122         p++;
00123         out[i] = (uint8_t)((hi « 4U) | lo);
00124     }
00125 }
00126
00140 static void build_usdc_calldata(uint8_t out[68], const char *to_hex, uint64_t amount)
00141 {
00142     (void)memset(out, 0, 68U);
00143
00144     /* Function selector */
00145     (void)memcpy(out, TRANSFER_SELECTOR, 4U);
00146
00147     /* 'to' address: right-aligned in 32-byte slot starting at offset 4 */
00148     uint8_t addr[20];
00149     parse_address(to_hex, addr);
00150     (void)memcpy(out + 4U + 12U, addr, 20U);
00151
00152     /* amount: right-aligned uint256, 64-bit value fits in the last 8 bytes */
00153     size_t j;
00154     for (j = 0U; j < 8U; j++) {
00155         out[67U - j] = (uint8_t)((amount » (8U * j)) & 0xFFU);
00156     }
00157 }
00158
00159 /**********************************************************************
00160  * Signing loop
00161  **********************************************************************/
00162
00173 static void signing_loop(CryptnoxWallet &wallet)
00174 {
00175     /* CARD_PIN is a string literal ("000000000"); copy into the pin array. */
00176     uint8_t card_pin[CW_MAX_PIN_LENGTH] = {};
00177     const size_t pin_len = (CARD_PIN_LEN < CW_MAX_PIN_LENGTH) ? CARD_PIN_LEN
00178                                                               : CW_MAX_PIN_LENGTH;
00179     (void)CW_Utils::safe_memcpy(card_pin, sizeof(card_pin),
00180                                 reinterpret_cast<const uint8_t *>(CARD_PIN),
00181                                 pin_len);
00182
00183     uint8_t calldata[68];
00184     build_usdc_calldata(calldata, "0x" ADDR_TO, AMOUNT_USDC);
00185
00186     while (true) {
00187         /* - 1. Wait for card ----------------------- */
00188         ESP_LOGI(TAG, "Hold Cryptnox card to reader to sign...");
```

```
00189
00190         CW_SecureSession session;
00191         bool connected = wallet.connect(session);
00192         if (!connected) {
00193             vTaskDelay(pdMS_TO_TICKS(100U));
00194             continue;
00195         }
00196
00197         /* - 2. Get fresh nonce (card is present -- fetch now) --- */
00198         uint64_t nonce = 0U;
00199         if (!eth_rpc_get_nonce(&nonce)) {
00200             ESP_LOGE(TAG, "Failed to get nonce -- retrying in 5 s");
00201             wallet.disconnect(session);
00202             vTaskDelay(pdMS_TO_TICKS(5000U));
00203             continue;
00204         }
00205
00206         /* - 3. Build tx --------------------------- */
00207         eth_tx_t tx = {};
00208         tx.chain_id          = CHAIN_ID_SEPOLIA;
00209         tx.nonce             = nonce;
00210         tx.max_priority_fee  = MAX_PRIORITY_FEE;
00211         tx.max_fee           = MAX_FEE;
00212         tx.gas_limit         = GAS_LIMIT_ERC20;
00213         tx.eth_value         = 0U;
00214         tx.calldata          = calldata;
00215         tx.calldata_len      = sizeof(calldata);
00216         parse_address("0x" ADDR_USDC, tx.to);
00217
00218         /* - 4. Encode unsigned tx and hash it ------------- */
00219         uint8_t unsigned_tx[TX_BUF_SIZE];
00220         size_t  unsigned_len = eth_rlp_encode_unsigned(&tx, unsigned_tx, sizeof(unsigned_tx));
00221         if (unsigned_len == 0U) {
00222             ESP_LOGE(TAG, "RLP encode unsigned overflow");
00223             wallet.disconnect(session);
00224             vTaskDelay(pdMS_TO_TICKS(2000U));
00225             continue;
00226         }
00227
00228         uint8_t hash[CW_HASH_SIZE];
00229         keccak256(unsigned_tx, unsigned_len, hash);
00230
00231         ESP_LOGI(TAG, "Hash to sign:");
00232         ESP_LOG_BUFFER_HEX_LEVEL(TAG, hash, CW_HASH_SIZE, ESP_LOG_INFO);
00233
00234         /* BIP32 Ethereum derivation path: m/44'/60'/0'/0/0
00235          * Each level is a 4-byte big-endian uint32; hardened levels have the
00236          * high bit set. */
00237         static const uint8_t eth_path[20] = {
00238             0x80U, 0x00U, 0x00U, 0x2CU,   /* 44' */
00239             0x80U, 0x00U, 0x00U, 0x3CU,   /* 60' */
00240             0x80U, 0x00U, 0x00U, 0x00U,   /* 0'  */
00241             0x00U, 0x00U, 0x00U, 0x00U,   /* 0   */
00242             0x00U, 0x00U, 0x00U, 0x00U,   /* 0   */
00243         };
00244
00245         CW_SignRequest req(session,
00246                            CW_SIGN_DERIVE_K1,
00247                            CW_SIGN_SIG_ECDSA_LOW_S,
00248                            CW_SIGN_WITH_PIN);
00249         req.hash             = hash;
00250         req.hashLength       = static_cast<uint8_t>(CW_HASH_SIZE);
00251         req.derivePath       = eth_path;
00252         req.derivePathLength = static_cast<uint8_t>(sizeof(eth_path));
00253         (void)CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
00254                                     card_pin, CW_MAX_PIN_LENGTH);
00255
00256         CW_SignResult result = wallet.sign(req);
00257         wallet.disconnect(session);
00258
00259         if (result.errorCode != CW_OK) {
00260             ESP_LOGE(TAG, "Sign failed: 0x%02X",
00261                      static_cast<unsigned int>(result.errorCode));
00262             vTaskDelay(pdMS_TO_TICKS(2000U));
00263             continue;
00264         }
00265
00266         uint8_t sig_r[CW_HASH_SIZE];
00267         uint8_t sig_s[CW_HASH_SIZE];
00268         (void)CW_Utils::safe_memcpy(sig_r, sizeof(sig_r),
00269                                     result.signature + CW_SIG_R_OFFSET, CW_HASH_SIZE);
00270         (void)CW_Utils::safe_memcpy(sig_s, sizeof(sig_s),
00271                                     result.signature + CW_SIG_S_OFFSET, CW_HASH_SIZE);
00272
00273         ESP_LOGI(TAG, "r:");
00274         ESP_LOG_BUFFER_HEX_LEVEL(TAG, sig_r, CW_HASH_SIZE, ESP_LOG_INFO);
00275         ESP_LOGI(TAG, "s:");
```

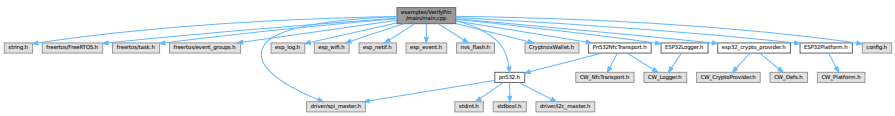
```
00276         ESP_LOG_BUFFER_HEX_LEVEL(TAG, sig_s, CW_HASH_SIZE, ESP_LOG_INFO);
00277
00278         /* - 6. Determine v parity via ecrecover ----------- */
00279         uint8_t v = eth_rpc_ecrecover_parity(hash, sig_r, sig_s);
00280         ESP_LOGI(TAG, "v = %u", static_cast<unsigned int>(v));
00281
00282         /* - 7. Encode signed tx ---------------------- */
00283         uint8_t signed_tx[TX_BUF_SIZE];
00284         size_t  signed_len = eth_rlp_encode_signed(&tx, v, sig_r, sig_s,
00285                                                    signed_tx, sizeof(signed_tx));
00286         if (signed_len == 0U) {
00287             ESP_LOGE(TAG, "RLP encode signed overflow");
00288             vTaskDelay(pdMS_TO_TICKS(2000U));
00289             continue;
00290         }
00291
00292         ESP_LOGI(TAG, "Signed tx (%u bytes):", (unsigned int)signed_len);
00293         ESP_LOG_BUFFER_HEX_LEVEL(TAG, signed_tx, signed_len, ESP_LOG_INFO);
00294
00295         /* - 8. Broadcast ---------------------------- */
00296         char tx_hash[68] = { 0 };
00297         if (eth_rpc_send_raw_tx(signed_tx, signed_len,
00298                                 tx_hash, sizeof(tx_hash))) {
00299             ESP_LOGI(TAG, "TX broadcast OK: %s", tx_hash);
00300         } else {
00301             ESP_LOGE(TAG, "TX broadcast failed");
00302         }
00303
00304         /* Wait before next iteration so nonce advances on-chain. */
00305         vTaskDelay(pdMS_TO_TICKS(15000U));
00306     }
00307 }
00308
00309 /*****************************************************************
00310  * Entry point
00311  *****************************************************************/
00312
00319 extern "C" void app_main(void)
00320 {
00321     /* - NVS (required by WiFi driver) ----------------- */
00322     esp_err_t nvs_ret = nvs_flash_init();
00323     if ((nvs_ret == ESP_ERR_NVS_NO_FREE_PAGES) ||
00324         (nvs_ret == ESP_ERR_NVS_NEW_VERSION_FOUND)) {
00325         ESP_ERROR_CHECK(nvs_flash_erase());
00326         nvs_ret = nvs_flash_init();
00327     }
00328     ESP_ERROR_CHECK(nvs_ret);
00329
00330     /* - PN532 NFC reader (transport selected at the top of this file) - */
00331     pn532_t nfc = {};
00332     pn532_config_t nfc_cfg = {};
00333 #if SPI_ENABLED
00334     spi_bus_config_t buscfg = {};
00335     buscfg.mosi_io_num     = SPI_MOSI;
00336     buscfg.miso_io_num     = SPI_MISO;
00337     buscfg.sclk_io_num     = SPI_SCLK;
00338     buscfg.quadwp_io_num   = SPI_PIN_UNUSED;
00339     buscfg.quadhd_io_num   = SPI_PIN_UNUSED;
00340     buscfg.max_transfer_sz = SPI_MAX_TRANSFER_SZ;
00341     ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &buscfg, SPI_DMA_CH_AUTO));
00342
00343     nfc_cfg.transport     = PN532_TRANSPORT_SPI;
00344     nfc_cfg.spi_host      = SPI2_HOST;
00345     nfc_cfg.pin_cs        = NFC_CS;
00346     nfc_cfg.skip_bus_init = true;
00347 #endif
00348
00349 #if I2C_ENABLED
00350     nfc_cfg.transport     = PN532_TRANSPORT_I2C;
00351     nfc_cfg.i2c_port      = PN532_I2C_PORT;
00352     nfc_cfg.pin_sda       = PN532_SDA;
00353     nfc_cfg.pin_scl       = PN532_SCL;
00354     nfc_cfg.pin_irq       = PN532_IRQ;
00355     nfc_cfg.pin_rst       = PN532_RST;
00356     nfc_cfg.i2c_clock_hz  = PN532_I2C_HZ;
00357 #endif
00358
00359     esp_err_t nfc_ret = pn532_init(&nfc, &nfc_cfg);
00360     if (nfc_ret != ESP_OK) {
00361         ESP_LOGE(TAG, "PN532 init failed");
00362         return;
00363     }
00364
00365     /* - Wallet setup ----------------------------- */
00366     ESP32Logger logger;
00367     (void)logger.begin(115200UL);
00368
```

```
00369     ESP32CryptoProvider cryptoProvider;
00370     ESP32Platform       platform;
00371     Pn532NfcTransport   nfcTransport(&nfc, logger);
00372     CryptnoxWallet      wallet(nfcTransport, logger, cryptoProvider, platform);
00373 
00374     if (!wallet.begin()) {
00375         ESP_LOGE(TAG, "Wallet begin (SAMConfig) failed");
00376         return;
00377     }
00378     /* wallet.begin() already prints the PN532 firmware version internally. */
00379 
00380     /* - WiFi + RPC ------------------------------ */
00381     eth_rpc_init(RPC_URL, "0x" ADDR_FROM);
00382 #if defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET)
00383     eth_rpc_set_auth(RPC_PROJECT_ID, RPC_API_SECRET);
00384 #endif
00385 
00386     if (!eth_rpc_wifi_connect(WIFI_SSID, WIFI_PASSWORD)) {
00387         ESP_LOGE(TAG, "WiFi connect failed -- check config.h credentials");
00388         return;
00389     }
00390 
00391     ESP_LOGI(TAG, "Ready -- will sign USDC transfer each card tap");
00392 
00393     signing_loop(wallet);
00394 }
```

5.72 examples/VerifyPin/main/main.cpp File Reference

```
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "driver/spi_master.h"
#include "esp_log.h"
#include "esp_wifi.h"
#include "esp_netif.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "pn532.h"
#include "CryptnoxWallet.h"
#include "Pn532NfcTransport.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "ESP32Platform.h"
#include "config.h"
```

Include dependency graph for main.cpp:



Macros

- #define SPI_MOSI 11
- #define SPI_MISO 13
- #define SPI_SCLK 12
- #define SPI_MAX_TRANSFER_SZ 4096
- #define SPI_PIN_UNUSED (-1)
- #define NFC_CS 10
- #define WIFI_CONNECTED_BIT BIT0
- #define WIFI_FAIL_BIT BIT1

Functions

- static void wifi_event_handler (void *arg, esp_event_base_t event_base, int32_t event_id, void *event_data)

  FreeRTOS event handler driving the Wi-Fi station state machine.
- static void wifi_start (void)

  Initialise Wi-Fi station mode and block until connected or timeout.
- static void run_verify_pin_loop (CryptnoxWallet &wallet)

  Main application loop: connect, verify PIN, and disconnect.
- void app_main (void)

  ESP-IDF application entry point.

Variables

- static const char *const TAG = "verify_pin"
- static const uint32_t LOOP_DELAY_MS = 1000U
- static const uint32_t WIFI_TIMEOUT_MS = 10000U
- static const int WIFI_MAX_RETRY = 5
- static EventGroupHandle_t s_wifi_event_group
- static int s_retry_num = 0

5.72.1 Macro Definition Documentation

5.72.1.1 NFC_CS

```
#define NFC_CS 10
```

Definition at line 53 of file main.cpp.

5.72.1.2 SPI_MAX_TRANSFER_SZ

```
#define SPI_MAX_TRANSFER_SZ 4096
```

Definition at line 51 of file main.cpp.

5.72.1.3 SPI_MISO

```
#define SPI_MISO 13
```

Definition at line 49 of file main.cpp.

5.72.1.4 SPI_MOSI

```
#define SPI_MOSI 11
```

Definition at line 48 of file main.cpp.

5.72.1.5 SPI_PIN_UNUSED

```
#define SPI_PIN_UNUSED (-1)
```

Definition at line 52 of file main.cpp.

5.72.1.6 SPI_SCLK

```
#define SPI_SCLK 12
```

Definition at line 50 of file main.cpp.

5.72.1.7 WIFI_CONNECTED_BIT

```
#define WIFI_CONNECTED_BIT BIT0
```

Definition at line 60 of file main.cpp.

5.72.1.8 WIFI_FAIL_BIT

```
#define WIFI_FAIL_BIT BIT1
```

Definition at line 61 of file main.cpp.

5.72.2 Function Documentation

5.72.2.1 app_main()

```
void app_main (
            void )
```

ESP-IDF application entry point.
Initialises NVS, starts Wi-Fi for full TRNG entropy, brings up the SPI bus and PN532 reader, then enters run_verify_pin_loop.
Definition at line 210 of file main.cpp.
References ESP32Logger::begin(), NFC_CS, pn532_config_t::pin_cs, pn532_init(), run_verify_pin_loop(), pn532_config_t::skip_bus_init, pn532_config_t::spi_host, SPI_MAX_TRANSFER_SZ, SPI_MISO, SPI_MOSI, SPI_PIN_UNUSED, SPI_SCLK, TAG, and wifi_start().

5.72.2.2 run_verify_pin_loop()

```
void run_verify_pin_loop (
            CryptnoxWallet & wallet)  [static]
```

Main application loop: connect, verify PIN, and disconnect.
Each iteration establishes the secure channel and submits the PIN via CryptnoxWallet::verifyPin. Halts permanently on rejection to protect the card's retry counter.

Parameters

in	*wallet*	Initialised wallet instance.

Examples

VerifyPin/main/main.cpp.

Definition at line 161 of file main.cpp.
References LOOP_DELAY_MS, and TAG.
Referenced by app_main().

5.72.2.3 wifi_event_handler()

```
void wifi_event_handler (
            void * arg,
            esp_event_base_t event_base,
            int32_t event_id,
            void * event_data)  [static]
```

FreeRTOS event handler driving the Wi-Fi station state machine.
Handles `WIFI_EVENT_STA_START` (triggers connection), `WIFI_EVENT_STA_DISCONNECTED` (retries up to `WIFI_MAX_RETRY` times), and `IP_EVENT_STA_GOT_IP` (signals success via the event group).

Parameters

in	*arg*	Unused.
in	*event_base*	Event base (`WIFI_EVENT` or `IP_EVENT`).
in	*event_id*	Event identifier within `event_base`.
in	*event_data*	Event-specific data (unused).

Definition at line 78 of file main.cpp.
References s_retry_num, s_wifi_event_group, WIFI_CONNECTED_BIT, WIFI_FAIL_BIT, and WIFI_MAX_RETRY.

5.72.2.4 wifi_start()

```
void wifi_start (
            void )  [static]
```

Initialise Wi-Fi station mode and block until connected or timeout.
Starts the ESP Wi-Fi stack, registers wifi_event_handler, configures the SSID and password from WIFI_SSID / WIFI_PASSWORD in `config.h`, then waits up to `WIFI_TIMEOUT_MS` for an IP address. The radio must be active before crypto operations so the hardware TRNG operates with full entropy (SEC-001).

Note

If the connection fails the function logs a warning and returns normally; the firmware continues with reduced TRNG entropy.

Definition at line 113 of file main.cpp.
References s_retry_num, s_wifi_event_group, TAG, WIFI_CONNECTED_BIT, wifi_event_handler(), WIFI_FAIL_BIT, WIFI_PASSWORD, WIFI_SSID, and WIFI_TIMEOUT_MS.

5.72.3 Variable Documentation

5.72.3.1 LOOP_DELAY_MS

```
const uint32_t LOOP_DELAY_MS = 1000U  [static]
```

Definition at line 56 of file main.cpp.

5.72.3.2 s_retry_num

```
int s_retry_num = 0  [static]
```

Definition at line 64 of file main.cpp.

5.72.3.3 s_wifi_event_group

```
EventGroupHandle_t s_wifi_event_group  [static]
```

Definition at line 63 of file main.cpp.

5.72.3.4 TAG

```
const char* const TAG = "verify_pin"  [static]
```

Definition at line 55 of file main.cpp.

5.72.3.5 WIFI_MAX_RETRY

```
const int WIFI_MAX_RETRY = 5  [static]
```

Definition at line 58 of file main.cpp.

5.72.3.6 WIFI_TIMEOUT_MS

```
const uint32_t WIFI_TIMEOUT_MS = 10000U  [static]
```

Definition at line 57 of file main.cpp.

5.73 main.cpp

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00028
00029 #include <string.h>
00030 #include "freertos/FreeRTOS.h"
00031 #include "freertos/task.h"
00032 #include "freertos/event_groups.h"
00033 #include "driver/spi_master.h"
00034 #include "esp_log.h"
00035 #include "esp_wifi.h"
00036 #include "esp_netif.h"
00037 #include "esp_event.h"
00038 #include "nvs_flash.h"
00039 #include "pn532.h"
00040 #include "CryptnoxWallet.h"
00041 #include "Pn532NfcTransport.h"
00042 #include "ESP32Logger.h"
00043 #include "esp32_crypto_provider.h"
00044 #include "ESP32Platform.h"
00045 #include "config.h"
00046
00047 /* - SPI wiring -- ESP32-S3 dev kit + Keyestudio PN532 breakout --- */
00048 #define SPI_MOSI            11
00049 #define SPI_MISO            13
00050 #define SPI_SCLK            12
00051 #define SPI_MAX_TRANSFER_SZ 4096
00052 #define SPI_PIN_UNUSED      (-1)
00053 #define NFC_CS              10
00054
00055 static const char *const TAG           = "verify_pin";
00056 static const uint32_t    LOOP_DELAY_MS  = 1000U;
00057 static const uint32_t    WIFI_TIMEOUT_MS = 10000U;
00058 static const int         WIFI_MAX_RETRY  = 5;
00059
00060 #define WIFI_CONNECTED_BIT BIT0
00061 #define WIFI_FAIL_BIT      BIT1
00062
00063 static EventGroupHandle_t s_wifi_event_group;
00064 static int                s_retry_num = 0;
00065
00078 static void wifi_event_handler(void *arg, esp_event_base_t event_base,
00079                                int32_t event_id, void *event_data)
00080 {
00081     (void)arg;
00082     (void)event_data;
00083     if ((event_base == WIFI_EVENT) && (event_id == WIFI_EVENT_STA_START)) {
00084         esp_wifi_connect();
00085     } else if ((event_base == WIFI_EVENT) &&
00086                (event_id == WIFI_EVENT_STA_DISCONNECTED)) {
00087         if (s_retry_num < WIFI_MAX_RETRY) {
00088             esp_wifi_connect();
00089             s_retry_num++;
00090         } else {
00091             xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
00092         }
00093     } else if ((event_base == IP_EVENT) && (event_id == IP_EVENT_STA_GOT_IP)) {
00094         s_retry_num = 0;
00095         xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
00096     } else {
00097         /* other events ignored */
00098     }
00099 }
00100
00113 static void wifi_start(void)
00114 {
00115     s_wifi_event_group = xEventGroupCreate();
00116     s_retry_num = 0;
00117     ESP_ERROR_CHECK(esp_netif_init());
00118     ESP_ERROR_CHECK(esp_event_loop_create_default());
00119     (void)esp_netif_create_default_wifi_sta();
00120     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
00121     ESP_ERROR_CHECK(esp_wifi_init(&cfg));
00122     esp_event_handler_instance_t h_any;
00123     esp_event_handler_instance_t h_ip;
00124     ESP_ERROR_CHECK(esp_event_handler_instance_register(
00125         WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler, NULL, &h_any));
00126     ESP_ERROR_CHECK(esp_event_handler_instance_register(
00127         IP_EVENT, IP_EVENT_STA_GOT_IP, &wifi_event_handler, NULL, &h_ip));
00128     wifi_config_t wifi_cfg;
00129     (void)memset(&wifi_cfg, 0, sizeof(wifi_cfg));
```

```
00130     (void)strncpy((char *)wifi_cfg.sta.ssid,     WIFI_SSID,
00131                   sizeof(wifi_cfg.sta.ssid)     - 1U);
00132     (void)strncpy((char *)wifi_cfg.sta.password, WIFI_PASSWORD,
00133                   sizeof(wifi_cfg.sta.password) - 1U);
00134     wifi_cfg.sta.threshold.authmode = WIFI_AUTH_WPA2_PSK;
00135     ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
00136     ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_cfg));
00137     ESP_ERROR_CHECK(esp_wifi_start());
00138     EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
00139                                            WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
00140                                            pdFALSE, pdFALSE,
00141                                            pdMS_TO_TICKS(WIFI_TIMEOUT_MS));
00142     if ((bits & WIFI_CONNECTED_BIT) != 0U) {
00143         ESP_LOGI(TAG, "WiFi connected");
00144     } else {
00145         ESP_LOGW(TAG, "WiFi connect failed -- TRNG entropy may be reduced");
00146     }
00147     (void)esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP, h_ip);
00148     (void)esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID, h_any);
00149     vEventGroupDelete(s_wifi_event_group);
00150 }
00151
00161 static void run_verify_pin_loop(CryptnoxWallet &wallet)
00162 {
00163     /* Replace with the PIN set on the card (4-9 ASCII digits). */
00164     static const uint8_t DEMO_PIN[CW_MAX_PIN_LENGTH] = {
00165         '0', '0', '0', '0', '0', '0', '0', '0', '0'
00166     };
00167
00168     bool keep_running = true;
00169     while (keep_running) {
00170         CW_SecureSession session{};
00171         bool connected = wallet.connect(session);
00172
00173         if (!connected) {
00174             ESP_LOGW(TAG, "Card not detected");
00175         }
00176
00177         if (connected) {
00178             bool pinOk = wallet.verifyPin(session,
00179                                           DEMO_PIN,
00180                                           static_cast<uint8_t>(CW_MAX_PIN_LENGTH));
00181             if (pinOk) {
00182                 ESP_LOGI(TAG, "PIN accepted");
00183             } else {
00184                 /* CRITICAL: do NOT retry -- each wrong PIN decrements the card's
00185                  * retry counter. Reaching 0 permanently blocks the PIN.
00186                  * Fix DEMO_PIN before flashing again. */
00187                 ESP_LOGE(TAG, "PIN rejected -- halting to protect retry counter");
00188                 keep_running = false;
00189             }
00190         }
00191
00192         wallet.disconnect(session);
00193
00194         if (keep_running) {
00195             vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
00196         }
00197     }
00198
00199     while (true) {
00200         vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
00201     }
00202 }
00203
00210 extern "C" void app_main(void)
00211 {
00212     esp_err_t nvs_ret = nvs_flash_init();
00213     if ((nvs_ret == ESP_ERR_NVS_NO_FREE_PAGES) ||
00214         (nvs_ret == ESP_ERR_NVS_NEW_VERSION_FOUND)) {
00215         ESP_ERROR_CHECK(nvs_flash_erase());
00216         nvs_ret = nvs_flash_init();
00217     }
00218     ESP_ERROR_CHECK(nvs_ret);
00219     wifi_start();
00220
00221     spi_bus_config_t buscfg = {};
00222     buscfg.mosi_io_num     = SPI_MOSI;
00223     buscfg.miso_io_num     = SPI_MISO;
00224     buscfg.sclk_io_num     = SPI_SCLK;
00225     buscfg.quadwp_io_num   = SPI_PIN_UNUSED;
00226     buscfg.quadhd_io_num   = SPI_PIN_UNUSED;
00227     buscfg.max_transfer_sz = SPI_MAX_TRANSFER_SZ;
00228     ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &buscfg, SPI_DMA_CH_AUTO));
00229
00230     pn532_t nfc = {};
00231     pn532_config_t nfc_cfg = {};
```

```
00232     nfc_cfg.spi_host      = SPI2_HOST;
00233     nfc_cfg.pin_cs        = NFC_CS;
00234     nfc_cfg.skip_bus_init = true;
00235     esp_err_t nfc_ret = pn532_init(&nfc, &nfc_cfg);
00236
00237     if (nfc_ret == ESP_OK) {
00238         ESP32Logger logger;
00239         (void)logger.begin(115200UL);
00240
00241         ESP32CryptoProvider cryptoProvider;
00242         ESP32Platform platform;
00243         Pn532NfcTransport nfcTransport(&nfc, logger);
00244         CryptnoxWallet wallet(nfcTransport, logger, cryptoProvider, platform);
00245
00246         if (wallet.begin()) {
00247             run_verify_pin_loop(wallet);
00248         } else {
00249             ESP_LOGE(TAG, "Wallet init failed");
00250         }
00251     } else {
00252         ESP_LOGE(TAG, "PN532 init failed");
00253     }
00254 }
```

Chapter 6

Examples

6.1 BasicUsage/main/main.cpp

BasicUsage ESP32 example: initialise PN532, connect to a Cryptnox card, and sign a hash.
BasicUsage ESP32 example: initialise PN532, connect to a Cryptnox card, and sign a hash.This is the ESP32-IDF port of the Arduino `BasicUsage.ino` sketch. It demonstrates the complete wallet flow with minimal code:

1. Initialise the PN532 NFC reader (SPI or I^2C — selected below).
2. Connect to a Cryptnox card and establish the secure channel.
3. Sign a 32-byte test hash and print the raw rs bytes.
4. Securely wipe sensitive buffers.

Select the communication interface by setting SPI_ENABLED / I2C_ENABLED at the top of this file.

Note

The card must have a seed loaded before signing will work. Use the Cryptnox CLI: `cryptnox seed generate`.

```
/*
 * SPDX-License-Identifier: LGPL-3.0-or-later
 * Copyright (c) 2026 Cryptnox SA
 */


#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "driver/spi_master.h"
#include "esp_log.h"
#include "esp_wifi.h"
#include "esp_netif.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "pn532.h"
#include "CryptnoxWallet.h"
#include "Pn532NfcTransport.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "ESP32Platform.h"
#include "CW_Utils.h"
#include "config.h"

/* ============================================================================
 * 1. Interface Selection
 * ----------------------------------------------------------------------------
 * Set exactly ONE flag to 1 and the other to 0.
 * ============================================================================ */
#define SPI_ENABLED         1
#define I2C_ENABLED         0

/* - SPI wiring -- ESP32-S3 dev kit + Keyestudio PN532 breakout --- */
#if SPI_ENABLED
#define SPI_MOSI            11
```

```
#define SPI_MISO            13
#define SPI_SCLK            12
#define SPI_MAX_TRANSFER_SZ 4096
#define SPI_PIN_UNUSED      (-1)
#define NFC_CS              10
#endif

/* - I²C wiring -- Cheap Yellow Display (ESP32) CN1 connector --- */
#if I2C_ENABLED
#define PN532_I2C_PORT      0          /* I2C_NUM_0 */
#define PN532_SDA           27         /* CN1 SDA */
#define PN532_SCL           22         /* CN1 SCL */
#define PN532_IRQ           (-1)       /* unused */
#define PN532_RST           (-1)       /* unused */
#define PN532_I2C_HZ        100000U
#endif

static const char *const TAG           = "basic_usage";
static const uint32_t    LOOP_DELAY_MS  = 1000U;
static const uint32_t    WIFI_TIMEOUT_MS = 10000U;
static const int         WIFI_MAX_RETRY  = 5;

#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT      BIT1

static EventGroupHandle_t s_wifi_event_group;
static int                s_retry_num = 0;

static void wifi_event_handler(void *arg, esp_event_base_t event_base,
                               int32_t event_id, void *event_data)
{
    (void)arg;
    (void)event_data;
    if ((event_base == WIFI_EVENT) && (event_id == WIFI_EVENT_STA_START)) {
        esp_wifi_connect();
    } else if ((event_base == WIFI_EVENT) &&
               (event_id == WIFI_EVENT_STA_DISCONNECTED)) {
        if (s_retry_num < WIFI_MAX_RETRY) {
            esp_wifi_connect();
            s_retry_num++;
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
    } else if ((event_base == IP_EVENT) && (event_id == IP_EVENT_STA_GOT_IP)) {
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    } else {
        /* other events ignored */
    }
}

static void wifi_start(void)
{
    s_wifi_event_group = xEventGroupCreate();
    s_retry_num = 0;
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    (void)esp_netif_create_default_wifi_sta();
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));
    esp_event_handler_instance_t h_any;
    esp_event_handler_instance_t h_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(
        WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler, NULL, &h_any));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(
        IP_EVENT, IP_EVENT_STA_GOT_IP, &wifi_event_handler, NULL, &h_ip));
    wifi_config_t wifi_cfg;
    (void)memset(&wifi_cfg, 0, sizeof(wifi_cfg));
    (void)strncpy((char *)wifi_cfg.sta.ssid,     WIFI_SSID,
                  sizeof(wifi_cfg.sta.ssid)     - 1U);
    (void)strncpy((char *)wifi_cfg.sta.password, WIFI_PASSWORD,
                  sizeof(wifi_cfg.sta.password) - 1U);
    wifi_cfg.sta.threshold.authmode = WIFI_AUTH_WPA2_PSK;
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_cfg));
    ESP_ERROR_CHECK(esp_wifi_start());
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
                                           WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
                                           pdFALSE, pdFALSE,
                                           pdMS_TO_TICKS(WIFI_TIMEOUT_MS));
    if ((bits & WIFI_CONNECTED_BIT) != 0U) {
        ESP_LOGI(TAG, "WiFi connected");
    } else {
        ESP_LOGW(TAG, "WiFi connect failed -- TRNG entropy may be reduced");
    }
    (void)esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP, h_ip);
    (void)esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID, h_any);
```

```
    vEventGroupDelete(s_wifi_event_group);
}

/* ============================================================================
 * 2. Default PIN
 * ----------------------------------------------------------------------------
 * Must match the PIN set on the card (4-9 ASCII digits).
 * ============================================================================ */
static const uint8_t DEFAULT_PIN[]    = "000000000";
static const size_t  DEFAULT_PIN_LEN  = sizeof(DEFAULT_PIN) - 1U;

/**********************************************************************
 * Main loop
 **********************************************************************/

static void run_basic_usage_loop(CryptnoxWallet &wallet)
{
    bool keep_running = true;

    while (keep_running) {
        /* - Step 1: Connect to card and establish secure channel - */
        CW_SecureSession session{};

        if (!wallet.connect(session)) {
            ESP_LOGW(TAG, "Card not detected -- hold card to reader");
            vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
            continue;
        }

        ESP_LOGI(TAG, "Card connected and secure channel established");

        /* - Step 2: Sign a test hash ------------------- */
        /* NOTE: Card must have a seed loaded before signing works.
         * Use the Cryptnox CLI: cryptnox seed generate */
        ESP_LOGI(TAG, "Signing test hash...");

        uint8_t testHash[CW_HASH_SIZE];
        (void)memset(testHash, 0x01, sizeof(testHash));

        /* Build sign request -- PIN included in sign payload for
         * authentication.  Alternatively call verifyPin() first
         * and use CW_SIGN_PINLESS_K1 here. */
        CW_SignRequest signRequest(session,
                                   CW_SIGN_CURR_K1,
                                   CW_SIGN_SIG_ECDSA_LOW_S,
                                   CW_SIGN_WITH_PIN);
        signRequest.hash       = testHash;
        signRequest.hashLength = static_cast<uint8_t>(CW_HASH_SIZE);
        (void)CW_Utils::safe_memcpy(signRequest.pin, sizeof(signRequest.pin),
                                    DEFAULT_PIN, DEFAULT_PIN_LEN);

        CW_SignResult signResult = wallet.sign(signRequest);

        if (signResult.errorCode == CW_OK) {
            ESP_LOGI(TAG, "Signature received (64 bytes raw r||s)");

            /* Print first 8 bytes of R for a quick visual check */
            ESP_LOG_BUFFER_HEX_LEVEL(TAG,
                &signResult.signature[CW_SIG_R_OFFSET], 8U, ESP_LOG_INFO);
            ESP_LOGI(TAG, "s:");
            ESP_LOG_BUFFER_HEX_LEVEL(TAG,
                &signResult.signature[CW_SIG_S_OFFSET], 8U, ESP_LOG_INFO);

            ESP_LOGI(TAG, "Card processed successfully");

        } else if (signResult.errorCode == CW_SIGN_PIN_INCORRECT) {
            /* CRITICAL: do NOT retry -- each wrong PIN decrements the
             * card's retry counter.  Reaching 0 permanently blocks
             * the PIN.  Fix DEFAULT_PIN before flashing again. */
            ESP_LOGE(TAG, "Wrong PIN -- halting to protect retry counter");
            CW_Utils::secure_wipe(testHash,            sizeof(testHash));
            CW_Utils::secure_wipe(signResult.signature, sizeof(signResult.signature));
            wallet.disconnect(session);
            keep_running = false;
            continue;

        } else {
            ESP_LOGE(TAG, "Sign failed: errorCode = 0x%02X",
                     static_cast<unsigned int>(signResult.errorCode));
        }

        /* - Step 3: Securely wipe sensitive buffers --------- */
        CW_Utils::secure_wipe(testHash,            sizeof(testHash));
        CW_Utils::secure_wipe(signResult.signature, sizeof(signResult.signature));

        /* Always disconnect to reset the reader for the next tap. */
```

```
        wallet.disconnect(session);

        vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
    }

    /* Halt after a fatal error (wrong PIN). */
    while (true) {
        vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
    }
}

/**************************************************************************
 * Entry point
 **************************************************************************/

extern "C" void app_main(void)
{
    esp_err_t nvs_ret = nvs_flash_init();
    if ((nvs_ret == ESP_ERR_NVS_NO_FREE_PAGES) ||
        (nvs_ret == ESP_ERR_NVS_NEW_VERSION_FOUND)) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        nvs_ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(nvs_ret);
    wifi_start();

    pn532_t        nfc     = {};
    pn532_config_t nfc_cfg = {};

#if SPI_ENABLED
    spi_bus_config_t buscfg = {};
    buscfg.mosi_io_num     = SPI_MOSI;
    buscfg.miso_io_num     = SPI_MISO;
    buscfg.sclk_io_num     = SPI_SCLK;
    buscfg.quadwp_io_num   = SPI_PIN_UNUSED;
    buscfg.quadhd_io_num   = SPI_PIN_UNUSED;
    buscfg.max_transfer_sz = SPI_MAX_TRANSFER_SZ;
    ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &buscfg, SPI_DMA_CH_AUTO));

    nfc_cfg.transport     = PN532_TRANSPORT_SPI;
    nfc_cfg.spi_host      = SPI2_HOST;
    nfc_cfg.pin_cs        = NFC_CS;
    nfc_cfg.skip_bus_init = true;
#endif

#if I2C_ENABLED
    nfc_cfg.transport    = PN532_TRANSPORT_I2C;
    nfc_cfg.i2c_port     = PN532_I2C_PORT;
    nfc_cfg.pin_sda      = PN532_SDA;
    nfc_cfg.pin_scl      = PN532_SCL;
    nfc_cfg.pin_irq      = PN532_IRQ;
    nfc_cfg.pin_rst      = PN532_RST;
    nfc_cfg.i2c_clock_hz = PN532_I2C_HZ;
#endif

    esp_err_t nfc_ret = pn532_init(&nfc, &nfc_cfg);
    if (nfc_ret != ESP_OK) {
        ESP_LOGE(TAG, "PN532 init failed -- check wiring and interface selection");
        return;
    }

    ESP32Logger          logger;
    (void)logger.begin(115200UL);

    ESP32CryptoProvider  cryptoProvider;
    ESP32Platform        platform;
    Pn532NfcTransport    nfcTransport(&nfc, logger);
    CryptnoxWallet       wallet(nfcTransport, logger, cryptoProvider, platform);

    if (!wallet.begin()) {
        ESP_LOGE(TAG, "Wallet init failed (SAMConfig)");
        return;
    }

    /* Print PN532 firmware version as a sanity check. */
    (void)nfcTransport.printFirmwareVersion();

    ESP_LOGI(TAG, "Ready -- hold Cryptnox card to reader");

    run_basic_usage_loop(wallet);
}
```

6.2 Connect/main/main.cpp

Minimal Cryptnox ESP32 example: open a secure channel and fetch card info.
Minimal Cryptnox ESP32 example: open a secure channel and fetch card info.Wiring & prerequisites:

- PN532 NFC reader on SPI: MOSI=11, MISO=13, SCLK=12, CS=10.
- A Cryptnox card initialised (use `cryptnox initialize`).
- `config.h` filled in with WIFI_SSID and WIFI_PASSWORD.

What the firmware does in each loop iteration:

1. Connect to the card and establish the secure channel (CryptnoxWallet::connect).
2. On success, fetch card owner info (CryptnoxWallet::getCardInfo).
3. Disconnect (CryptnoxWallet::disconnect).

This example never submits a PIN, so it cannot lock the card. It is the safest starting point to validate wiring and the secure channel before moving to the `VerifyPin` or `Sign` examples.

```
/*
 * SPDX-License-Identifier: LGPL-3.0-or-later
 * Copyright (c) 2026 Cryptnox SA
 */


#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "driver/spi_master.h"
#include "esp_log.h"
#include "esp_wifi.h"
#include "esp_netif.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "pn532.h"
#include "CryptnoxWallet.h"
#include "Pn532NfcTransport.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "ESP32Platform.h"
#include "config.h"

/* - SPI wiring -- ESP32-S3 dev kit + Keyestudio PN532 breakout --- */
#define SPI_MOSI            11
#define SPI_MISO            13
#define SPI_SCLK            12
#define SPI_MAX_TRANSFER_SZ 4096
#define SPI_PIN_UNUSED      (-1)
#define NFC_CS              10

static const char *const TAG           = "connect";
static const uint32_t    LOOP_DELAY_MS  = 1000U;
static const uint32_t    WIFI_TIMEOUT_MS = 10000U;
static const int         WIFI_MAX_RETRY  = 5;

#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT      BIT1

static EventGroupHandle_t s_wifi_event_group;
static int                s_retry_num = 0;

static void wifi_event_handler(void *arg, esp_event_base_t event_base,
                               int32_t event_id, void *event_data)
{
    (void)arg;
    (void)event_data;
    if ((event_base == WIFI_EVENT) && (event_id == WIFI_EVENT_STA_START)) {
        esp_wifi_connect();
    } else if ((event_base == WIFI_EVENT) &&
               (event_id == WIFI_EVENT_STA_DISCONNECTED)) {
        if (s_retry_num < WIFI_MAX_RETRY) {
            esp_wifi_connect();
            s_retry_num++;
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
    } else if ((event_base == IP_EVENT) && (event_id == IP_EVENT_STA_GOT_IP)) {
        s_retry_num = 0;
```

```
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    } else {
        /* other events ignored */
    }
}

static void wifi_start(void)
{
    s_wifi_event_group = xEventGroupCreate();
    s_retry_num = 0;
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    (void)esp_netif_create_default_wifi_sta();
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));
    esp_event_handler_instance_t h_any;
    esp_event_handler_instance_t h_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(
        WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler, NULL, &h_any));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(
        IP_EVENT, IP_EVENT_STA_GOT_IP, &wifi_event_handler, NULL, &h_ip));
    wifi_config_t wifi_cfg;
    (void)memset(&wifi_cfg, 0, sizeof(wifi_cfg));
    (void)strncpy((char *)wifi_cfg.sta.ssid,     WIFI_SSID,
                  sizeof(wifi_cfg.sta.ssid)     - 1U);
    (void)strncpy((char *)wifi_cfg.sta.password, WIFI_PASSWORD,
                  sizeof(wifi_cfg.sta.password) - 1U);
    wifi_cfg.sta.threshold.authmode = WIFI_AUTH_WPA2_PSK;
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_cfg));
    ESP_ERROR_CHECK(esp_wifi_start());
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
                                           WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
                                           pdFALSE, pdFALSE,
                                           pdMS_TO_TICKS(WIFI_TIMEOUT_MS));
    if ((bits & WIFI_CONNECTED_BIT) != 0U) {
        ESP_LOGI(TAG, "WiFi connected");
    } else {
        ESP_LOGW(TAG, "WiFi connect failed -- TRNG entropy may be reduced");
    }
    (void)esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP, h_ip);
    (void)esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID, h_any);
    vEventGroupDelete(s_wifi_event_group);
}

static void run_connect_loop(CryptnoxWallet &wallet)
{
    while (true) {
        CW_SecureSession session{};
        bool connected = wallet.connect(session);

        if (connected) {
            ESP_LOGI(TAG, "Card connected, secure channel established");

            CW_CardInfo info{};
            bool infoOk = wallet.getCardInfo(session, &info);
            if (infoOk) {
                ESP_LOGI(TAG, "Owner name : %s", info.name);
                ESP_LOGI(TAG, "Owner email: %s", info.email);
            } else {
                ESP_LOGW(TAG, "getCardInfo failed");
            }
        } else {
            ESP_LOGW(TAG, "Card not detected or secure channel failed");
        }

        wallet.disconnect(session);
        vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
    }
}

extern "C" void app_main(void)
{
    esp_err_t nvs_ret = nvs_flash_init();
    if ((nvs_ret == ESP_ERR_NVS_NO_FREE_PAGES) ||
        (nvs_ret == ESP_ERR_NVS_NEW_VERSION_FOUND)) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        nvs_ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(nvs_ret);
    wifi_start();

    spi_bus_config_t buscfg = {};
    buscfg.mosi_io_num     = SPI_MOSI;
    buscfg.miso_io_num     = SPI_MISO;
    buscfg.sclk_io_num     = SPI_SCLK;
    buscfg.quadwp_io_num   = SPI_PIN_UNUSED;
```

```
    buscfg.quadhd_io_num   = SPI_PIN_UNUSED;
    buscfg.max_transfer_sz = SPI_MAX_TRANSFER_SZ;
    ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &buscfg, SPI_DMA_CH_AUTO));

    pn532_t nfc = {};
    pn532_config_t nfc_cfg = {};
    nfc_cfg.spi_host      = SPI2_HOST;
    nfc_cfg.pin_cs        = NFC_CS;
    nfc_cfg.skip_bus_init = true;
    esp_err_t nfc_ret = pn532_init(&nfc, &nfc_cfg);

    if (nfc_ret == ESP_OK) {
        ESP32Logger logger;
        (void)logger.begin(115200UL);

        ESP32CryptoProvider cryptoProvider;
        ESP32Platform platform;
        Pn532NfcTransport nfcTransport(&nfc, logger);
        CryptnoxWallet wallet(nfcTransport, logger, cryptoProvider, platform);

        if (wallet.begin()) {
            (void)nfcTransport.printFirmwareVersion();
            run_connect_loop(wallet);
        } else {
            ESP_LOGE(TAG, "Wallet init failed");
        }
    } else {
        ESP_LOGE(TAG, "PN532 init failed");
    }
}
```

6.3 Sign/main/main.cpp

Minimal Cryptnox ESP32 example: sign a 32-byte hash on the secp256k1 curve.
Minimal Cryptnox ESP32 example: sign a 32-byte hash on the secp256k1 curve.Wiring & prerequisites:

- PN532 NFC reader on SPI: MOSI=11, MISO=13, SCLK=12, CS=10.
- A Cryptnox card initialised with a known PIN and a loaded seed (use the Cryptnox CLI↩ : `cryptnox initialize` then `cryptnox seed generate`).
- DEMO_PIN must match the PIN set on the card.
- `config.h` filled in with WIFI_SSID and WIFI_PASSWORD.

What the firmware does in each loop iteration:

1. Connect to the card and establish the secure channel.
2. Sign a 32-byte test hash on the secp256k1 curve (key type CW_SIGN_CURR_K1, signature type CW_SIGN_SIG_ECDSA_LOW_S, PIN included in the sign payload via CW_SIGN_WITH_PIN).
3. Print the raw rs signature bytes, wipe sensitive buffers, disconnect.

Warning

On CW_SIGN_PIN_INCORRECT the firmware enters an infinite halt: every wrong PIN attempt decrements the card's retry counter and reaching 0 permanently blocks the PIN. Verify DEMO_↩ PIN matches the card before flashing.

Note

The hash filled with 0x01 is a test pattern. In real use replace it with the SHA-256 (or Keccak-256 for Ethereum) digest of the transaction you want the card to sign.

```
/*
 * SPDX-License-Identifier: LGPL-3.0-or-later
 * Copyright (c) 2026 Cryptnox SA
 */


#include <string.h>
```

```
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "driver/spi_master.h"
#include "esp_log.h"
#include "esp_wifi.h"
#include "esp_netif.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "pn532.h"
#include "CryptnoxWallet.h"
#include "Pn532NfcTransport.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "ESP32Platform.h"
#include "config.h"

/* - SPI wiring -- ESP32-S3 dev kit + Keyestudio PN532 breakout --- */
#define SPI_MOSI            11
#define SPI_MISO            13
#define SPI_SCLK            12
#define SPI_MAX_TRANSFER_SZ 4096
#define SPI_PIN_UNUSED      (-1)
#define NFC_CS              10

static const char *const TAG           = "sign";
static const uint32_t    LOOP_DELAY_MS  = 1000U;
static const uint32_t    WIFI_TIMEOUT_MS = 10000U;
static const int         WIFI_MAX_RETRY  = 5;

#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT      BIT1

static EventGroupHandle_t s_wifi_event_group;
static int                s_retry_num = 0;

static void wifi_event_handler(void *arg, esp_event_base_t event_base,
                               int32_t event_id, void *event_data)
{
    (void)arg;
    (void)event_data;
    if ((event_base == WIFI_EVENT) && (event_id == WIFI_EVENT_STA_START)) {
        esp_wifi_connect();
    } else if ((event_base == WIFI_EVENT) &&
               (event_id == WIFI_EVENT_STA_DISCONNECTED)) {
        if (s_retry_num < WIFI_MAX_RETRY) {
            esp_wifi_connect();
            s_retry_num++;
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
    } else if ((event_base == IP_EVENT) && (event_id == IP_EVENT_STA_GOT_IP)) {
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    } else {
        /* other events ignored */
    }
}

static void wifi_start(void)
{
    s_wifi_event_group = xEventGroupCreate();
    s_retry_num = 0;
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    (void)esp_netif_create_default_wifi_sta();
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));
    esp_event_handler_instance_t h_any;
    esp_event_handler_instance_t h_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(
        WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler, NULL, &h_any));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(
        IP_EVENT, IP_EVENT_STA_GOT_IP, &wifi_event_handler, NULL, &h_ip));
    wifi_config_t wifi_cfg;
    (void)memset(&wifi_cfg, 0, sizeof(wifi_cfg));
    (void)strncpy((char *)wifi_cfg.sta.ssid,     WIFI_SSID,
                  sizeof(wifi_cfg.sta.ssid)     - 1U);
    (void)strncpy((char *)wifi_cfg.sta.password, WIFI_PASSWORD,
                  sizeof(wifi_cfg.sta.password) - 1U);
    wifi_cfg.sta.threshold.authmode = WIFI_AUTH_WPA2_PSK;
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_cfg));
    ESP_ERROR_CHECK(esp_wifi_start());
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
                                           WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
                                           pdFALSE, pdFALSE,
```

```
                                        pdMS_TO_TICKS(WIFI_TIMEOUT_MS));
    if ((bits & WIFI_CONNECTED_BIT) != 0U) {
        ESP_LOGI(TAG, "WiFi connected");
    } else {
        ESP_LOGW(TAG, "WiFi connect failed -- TRNG entropy may be reduced");
    }
    (void)esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP, h_ip);
    (void)esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID, h_any);
    vEventGroupDelete(s_wifi_event_group);
}

static void run_sign_loop(CryptnoxWallet &wallet)
{
    /* Replace with the SHA-256 (or Keccak-256) digest of the real transaction. */
    static const uint8_t TEST_HASH[CW_HASH_SIZE] = {
        0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U,
        0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U,
        0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U,
        0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U, 0x01U,
    };
    /* Replace with the PIN set on the card (4-9 ASCII digits). */
    static const uint8_t DEMO_PIN[CW_MAX_PIN_LENGTH] = {
        '0', '0', '0', '0', '0', '0', '0', '0', '0'
    };

    bool keep_running = true;
    while (keep_running) {
        CW_SecureSession session{};
        bool connected = wallet.connect(session);

        if (!connected) {
            ESP_LOGW(TAG, "Card not detected");
        }

        if (connected) {
            CW_SignRequest req(session,
                               CW_SIGN_CURR_K1,
                               CW_SIGN_SIG_ECDSA_LOW_S,
                               CW_SIGN_WITH_PIN);
            req.hash       = TEST_HASH;
            req.hashLength = static_cast<uint8_t>(CW_HASH_SIZE);
            (void)CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
                                        DEMO_PIN, CW_MAX_PIN_LENGTH);

            CW_SignResult result = wallet.sign(req);

            if (result.errorCode == CW_OK) {
                ESP_LOGI(TAG, "Sign OK - r:");
                ESP_LOG_BUFFER_HEX_LEVEL(TAG, &result.signature[CW_SIG_R_OFFSET],
                                         CW_HASH_SIZE, ESP_LOG_INFO);
                ESP_LOGI(TAG, "s:");
                ESP_LOG_BUFFER_HEX_LEVEL(TAG, &result.signature[CW_SIG_S_OFFSET],
                                         CW_HASH_SIZE, ESP_LOG_INFO);
            } else if (result.errorCode == CW_SIGN_PIN_INCORRECT) {
                /* CRITICAL: do NOT retry -- each wrong PIN decrements the card's
                 * retry counter. Reaching 0 permanently blocks the PIN.
                 * Fix DEMO_PIN before flashing again. */
                ESP_LOGE(TAG, "Wrong PIN -- halting to protect retry counter");
                keep_running = false;
            } else {
                ESP_LOGE(TAG, "Sign failed: 0x%02X",
                         static_cast<unsigned int>(result.errorCode));
            }
        }

        wallet.disconnect(session);

        if (keep_running) {
            vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
        }
    }

    while (true) {
        vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
    }
}

extern "C" void app_main(void)
{
    esp_err_t nvs_ret = nvs_flash_init();
    if ((nvs_ret == ESP_ERR_NVS_NO_FREE_PAGES) ||
        (nvs_ret == ESP_ERR_NVS_NEW_VERSION_FOUND)) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        nvs_ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(nvs_ret);
```

```
    wifi_start();

    spi_bus_config_t buscfg = {};
    buscfg.mosi_io_num     = SPI_MOSI;
    buscfg.miso_io_num     = SPI_MISO;
    buscfg.sclk_io_num     = SPI_SCLK;
    buscfg.quadwp_io_num   = SPI_PIN_UNUSED;
    buscfg.quadhd_io_num   = SPI_PIN_UNUSED;
    buscfg.max_transfer_sz = SPI_MAX_TRANSFER_SZ;
    ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &buscfg, SPI_DMA_CH_AUTO));

    pn532_t nfc = {};
    pn532_config_t nfc_cfg = {};
    nfc_cfg.spi_host      = SPI2_HOST;
    nfc_cfg.pin_cs        = NFC_CS;
    nfc_cfg.skip_bus_init = true;
    esp_err_t nfc_ret = pn532_init(&nfc, &nfc_cfg);

    if (nfc_ret == ESP_OK) {
        ESP32Logger logger;
        (void)logger.begin(115200UL);

        ESP32CryptoProvider cryptoProvider;
        ESP32Platform platform;
        Pn532NfcTransport nfcTransport(&nfc, logger);
        CryptnoxWallet wallet(nfcTransport, logger, cryptoProvider, platform);

        if (wallet.begin()) {
            run_sign_loop(wallet);
        } else {
            ESP_LOGE(TAG, "Wallet init failed");
        }
    } else {
        ESP_LOGE(TAG, "PN532 init failed");
    }
}
```

6.4 UsdcSigning/main/main.cpp

Cryptnox ESP32 example: build, sign, and broadcast a USDC ERC-20 transfer.
Cryptnox ESP32 example: build, sign, and broadcast a USDC ERC-20 transfer.Wiring & prerequisites:

- PN532 NFC reader — transport selected by PN532_USE_I2C at the top of this file.
- A Cryptnox card initialised with a seed and a known PIN.
- `config.h` filled in with WiFi, RPC endpoint, and Ethereum addresses (copy from `config.↩ template.h` and fill in the values).

What the firmware does in each loop iteration:

1. Wait for a card tap and establish the secure channel.
2. Fetch the current on-chain nonce from the RPC endpoint.
3. Build an EIP-1559 type-2 transaction for a USDC `transfer(address,uint256)` call.
4. RLP-encode the unsigned transaction and Keccak-256 hash it.
5. Sign the hash on the card (CryptnoxWallet::sign).
6. Recover the `v` parity bit via `eth_rpc_ecrecover_parity`.
7. RLP-encode the signed transaction and broadcast it.

Note

Fill in WIFI_SSID, WIFI_PASSWORD, RPC_URL, ADDR_FROM, ADDR_TO, ADDR_USDC, and CARD_PIN in `config.h` before building.

```
/*
 * SPDX-License-Identifier: LGPL-3.0-or-later
 * Copyright (c) 2026 Cryptnox SA
 */


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <inttypes.h>

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/spi_master.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "nvs_flash.h"

#include "CryptnoxWallet.h"
#include "CW_Utils.h"
#include "Pn532NfcTransport.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "ESP32Platform.h"

extern "C" {
#include "pn532.h"
#include "keccak256.h"
#include "eth_rlp.h"
#include "eth_rpc.h"
}

#include "config.h"

static const char *const TAG = "usdc_signing";

/* - PN532 transport selector -------------------------
 * Enable exactly one transport by setting its flag to 1 (the other to 0). */
#define SPI_ENABLED         1
#define I2C_ENABLED         0

/* - SPI wiring -- ESP32-S3 dev kit + Keyestudio PN532 breakout --- */
#if SPI_ENABLED
#define SPI_MOSI            11
#define SPI_MISO            13
#define SPI_SCLK            12
#define SPI_MAX_TRANSFER_SZ 4096
#define SPI_PIN_UNUSED      (-1)
#define NFC_CS              10
#endif

/* - I²C wiring -- Cheap Yellow Display (ESP32) CN1 connector --- */
#if I2C_ENABLED
#define PN532_I2C_PORT      0           /* I2C_NUM_0 */
#define PN532_SDA           27          /* CN1 SDA */
#define PN532_SCL           22          /* CN1 SCL */
#define PN532_IRQ           (-1)        /* unused */
#define PN532_RST           (-1)        /* unused */
#define PN532_I2C_HZ        100000U
#endif

/* - USDC ERC-20 transfer(address,uint256) selector -------- */
static const uint8_t TRANSFER_SELECTOR[4] = { 0xa9U, 0x05U, 0x9cU, 0xbbU };

/* - Unsigned and signed tx buffers (EIP-1559 type 2) ------- */
#define TX_BUF_SIZE 300U

/*****************************************************************
 * Helpers
 *****************************************************************/

static void parse_address(const char *hex, uint8_t out[20])
{
    const char *p = hex;
    if ((p[0] == '0') && ((p[1] == 'x') || (p[1] == 'X'))) {
        p += 2;
    }
    (void)memset(out, 0, 20U);
    size_t i;
    for (i = 0U; (i < 20U) && (p[0] != '\0') && (p[1] != '\0'); i++) {
        uint8_t hi = (uint8_t)((*p >= 'a') ? (*p - 'a' + 10) :
                               (*p >= 'A') ? (*p - 'A' + 10) : (*p - '0'));
```

```
        p++;
        uint8_t lo = (uint8_t)((*p >= 'a') ? (*p - 'a' + 10) :
                               (*p >= 'A') ? (*p - 'A' + 10) : (*p - '0'));
        p++;
        out[i] = (uint8_t)((hi << 4U) | lo);
    }
}

static void build_usdc_calldata(uint8_t out[68], const char *to_hex, uint64_t amount)
{
    (void)memset(out, 0, 68U);

    /* Function selector */
    (void)memcpy(out, TRANSFER_SELECTOR, 4U);

    /* 'to' address: right-aligned in 32-byte slot starting at offset 4 */
    uint8_t addr[20];
    parse_address(to_hex, addr);
    (void)memcpy(out + 4U + 12U, addr, 20U);

    /* amount: right-aligned uint256, 64-bit value fits in the last 8 bytes */
    size_t j;
    for (j = 0U; j < 8U; j++) {
        out[67U - j] = (uint8_t)((amount >> (8U * j)) & 0xFFU);
    }
}

/*****************************************************************
 * Signing loop
 *****************************************************************/

static void signing_loop(CryptnoxWallet &wallet)
{
    /* CARD_PIN is a string literal ("000000000"); copy into the pin array. */
    uint8_t card_pin[CW_MAX_PIN_LENGTH] = {};
    const size_t pin_len = (CARD_PIN_LEN < CW_MAX_PIN_LENGTH) ? CARD_PIN_LEN
                                                              : CW_MAX_PIN_LENGTH;
    (void)CW_Utils::safe_memcpy(card_pin, sizeof(card_pin),
                                reinterpret_cast<const uint8_t *>(CARD_PIN),
                                pin_len);

    uint8_t calldata[68];
    build_usdc_calldata(calldata, "0x" ADDR_TO, AMOUNT_USDC);

    while (true) {
        /* - 1. Wait for card ------------------------ */
        ESP_LOGI(TAG, "Hold Cryptnox card to reader to sign...");

        CW_SecureSession session;
        bool connected = wallet.connect(session);
        if (!connected) {
            vTaskDelay(pdMS_TO_TICKS(100U));
            continue;
        }

        /* - 2. Get fresh nonce (card is present -- fetch now) --- */
        uint64_t nonce = 0U;
        if (!eth_rpc_get_nonce(&nonce)) {
            ESP_LOGE(TAG, "Failed to get nonce -- retrying in 5 s");
            wallet.disconnect(session);
            vTaskDelay(pdMS_TO_TICKS(5000U));
            continue;
        }

        /* - 3. Build tx ---------------------------- */
        eth_tx_t tx = {};
        tx.chain_id          = CHAIN_ID_SEPOLIA;
        tx.nonce             = nonce;
        tx.max_priority_fee  = MAX_PRIORITY_FEE;
        tx.max_fee           = MAX_FEE;
        tx.gas_limit         = GAS_LIMIT_ERC20;
        tx.eth_value         = 0U;
        tx.calldata          = calldata;
        tx.calldata_len      = sizeof(calldata);
        parse_address("0x" ADDR_USDC, tx.to);

        /* - 4. Encode unsigned tx and hash it ------------- */
        uint8_t unsigned_tx[TX_BUF_SIZE];
        size_t  unsigned_len = eth_rlp_encode_unsigned(&tx, unsigned_tx, sizeof(unsigned_tx));
        if (unsigned_len == 0U) {
            ESP_LOGE(TAG, "RLP encode unsigned overflow");
            wallet.disconnect(session);
            vTaskDelay(pdMS_TO_TICKS(2000U));
            continue;
        }

        uint8_t hash[CW_HASH_SIZE];
```

```
        keccak256(unsigned_tx, unsigned_len, hash);

        ESP_LOGI(TAG, "Hash to sign:");
        ESP_LOG_BUFFER_HEX_LEVEL(TAG, hash, CW_HASH_SIZE, ESP_LOG_INFO);

        /* BIP32 Ethereum derivation path: m/44'/60'/0'/0/0
         * Each level is a 4-byte big-endian uint32; hardened levels have the
         * high bit set. */
        static const uint8_t eth_path[20] = {
            0x80U, 0x00U, 0x00U, 0x2CU,   /* 44' */
            0x80U, 0x00U, 0x00U, 0x3CU,   /* 60' */
            0x80U, 0x00U, 0x00U, 0x00U,   /* 0'  */
            0x00U, 0x00U, 0x00U, 0x00U,   /* 0   */
            0x00U, 0x00U, 0x00U, 0x00U,   /* 0   */
        };

        CW_SignRequest req(session,
                           CW_SIGN_DERIVE_K1,
                           CW_SIGN_SIG_ECDSA_LOW_S,
                           CW_SIGN_WITH_PIN);
        req.hash             = hash;
        req.hashLength       = static_cast<uint8_t>(CW_HASH_SIZE);
        req.derivePath       = eth_path;
        req.derivePathLength = static_cast<uint8_t>(sizeof(eth_path));
        (void)CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
                                    card_pin, CW_MAX_PIN_LENGTH);

        CW_SignResult result = wallet.sign(req);
        wallet.disconnect(session);

        if (result.errorCode != CW_OK) {
            ESP_LOGE(TAG, "Sign failed: 0x%02X",
                     static_cast<unsigned int>(result.errorCode));
            vTaskDelay(pdMS_TO_TICKS(2000U));
            continue;
        }

        uint8_t sig_r[CW_HASH_SIZE];
        uint8_t sig_s[CW_HASH_SIZE];
        (void)CW_Utils::safe_memcpy(sig_r, sizeof(sig_r),
                                    result.signature + CW_SIG_R_OFFSET, CW_HASH_SIZE);
        (void)CW_Utils::safe_memcpy(sig_s, sizeof(sig_s),
                                    result.signature + CW_SIG_S_OFFSET, CW_HASH_SIZE);

        ESP_LOGI(TAG, "r:");
        ESP_LOG_BUFFER_HEX_LEVEL(TAG, sig_r, CW_HASH_SIZE, ESP_LOG_INFO);
        ESP_LOGI(TAG, "s:");
        ESP_LOG_BUFFER_HEX_LEVEL(TAG, sig_s, CW_HASH_SIZE, ESP_LOG_INFO);

        /* - 6. Determine v parity via ecrecover ----------- */
        uint8_t v = eth_rpc_ecrecover_parity(hash, sig_r, sig_s);
        ESP_LOGI(TAG, "v = %u", static_cast<unsigned int>(v));

        /* - 7. Encode signed tx ---------------------- */
        uint8_t signed_tx[TX_BUF_SIZE];
        size_t  signed_len = eth_rlp_encode_signed(&tx, v, sig_r, sig_s,
                                                   signed_tx, sizeof(signed_tx));
        if (signed_len == 0U) {
            ESP_LOGE(TAG, "RLP encode signed overflow");
            vTaskDelay(pdMS_TO_TICKS(2000U));
            continue;
        }

        ESP_LOGI(TAG, "Signed tx (%u bytes):", (unsigned int)signed_len);
        ESP_LOG_BUFFER_HEX_LEVEL(TAG, signed_tx, signed_len, ESP_LOG_INFO);

        /* - 8. Broadcast ---------------------------- */
        char tx_hash[68] = { 0 };
        if (eth_rpc_send_raw_tx(signed_tx, signed_len,
                                tx_hash, sizeof(tx_hash))) {
            ESP_LOGI(TAG, "TX broadcast OK: %s", tx_hash);
        } else {
            ESP_LOGE(TAG, "TX broadcast failed");
        }

        /* Wait before next iteration so nonce advances on-chain. */
        vTaskDelay(pdMS_TO_TICKS(15000U));
    }
}

/*****************************************************************
 * Entry point
 *****************************************************************/

extern "C" void app_main(void)
{
    /* - NVS (required by WiFi driver) ----------------- */
```

```
    esp_err_t nvs_ret = nvs_flash_init();
    if ((nvs_ret == ESP_ERR_NVS_NO_FREE_PAGES) ||
        (nvs_ret == ESP_ERR_NVS_NEW_VERSION_FOUND)) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        nvs_ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(nvs_ret);

    /* - PN532 NFC reader (transport selected at the top of this file) - */
    pn532_t nfc = {};
    pn532_config_t nfc_cfg = {};
#if SPI_ENABLED
    spi_bus_config_t buscfg = {};
    buscfg.mosi_io_num     = SPI_MOSI;
    buscfg.miso_io_num     = SPI_MISO;
    buscfg.sclk_io_num     = SPI_SCLK;
    buscfg.quadwp_io_num   = SPI_PIN_UNUSED;
    buscfg.quadhd_io_num   = SPI_PIN_UNUSED;
    buscfg.max_transfer_sz = SPI_MAX_TRANSFER_SZ;
    ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &buscfg, SPI_DMA_CH_AUTO));

    nfc_cfg.transport     = PN532_TRANSPORT_SPI;
    nfc_cfg.spi_host      = SPI2_HOST;
    nfc_cfg.pin_cs        = NFC_CS;
    nfc_cfg.skip_bus_init = true;
#endif

#if I2C_ENABLED
    nfc_cfg.transport     = PN532_TRANSPORT_I2C;
    nfc_cfg.i2c_port      = PN532_I2C_PORT;
    nfc_cfg.pin_sda       = PN532_SDA;
    nfc_cfg.pin_scl       = PN532_SCL;
    nfc_cfg.pin_irq       = PN532_IRQ;
    nfc_cfg.pin_rst       = PN532_RST;
    nfc_cfg.i2c_clock_hz  = PN532_I2C_HZ;
#endif

    esp_err_t nfc_ret = pn532_init(&nfc, &nfc_cfg);
    if (nfc_ret != ESP_OK) {
        ESP_LOGE(TAG, "PN532 init failed");
        return;
    }

    /* - Wallet setup ------------------------------ */
    ESP32Logger logger;
    (void)logger.begin(115200UL);

    ESP32CryptoProvider cryptoProvider;
    ESP32Platform       platform;
    Pn532NfcTransport   nfcTransport(&nfc, logger);
    CryptnoxWallet      wallet(nfcTransport, logger, cryptoProvider, platform);

    if (!wallet.begin()) {
        ESP_LOGE(TAG, "Wallet begin (SAMConfig) failed");
        return;
    }
    /* wallet.begin() already prints the PN532 firmware version internally. */

    /* - WiFi + RPC -------------------------------- */
    eth_rpc_init(RPC_URL, "0x" ADDR_FROM);
#if defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET)
    eth_rpc_set_auth(RPC_PROJECT_ID, RPC_API_SECRET);
#endif

    if (!eth_rpc_wifi_connect(WIFI_SSID, WIFI_PASSWORD)) {
        ESP_LOGE(TAG, "WiFi connect failed -- check config.h credentials");
        return;
    }

    ESP_LOGI(TAG, "Ready -- will sign USDC transfer each card tap");

    signing_loop(wallet);
}
```

6.5 VerifyPin/main/main.cpp

Minimal Cryptnox ESP32 example: verify the card PIN over a secure channel.
Minimal Cryptnox ESP32 example: verify the card PIN over a secure channel.Wiring & prerequisites:

- PN532 NFC reader on SPI: MOSI=11, MISO=13, SCLK=12, CS=10.
- A Cryptnox card initialised with a known PIN.

- DEMO_PIN must match the PIN set on the card.
- `config.h` filled in with WIFI_SSID and WIFI_PASSWORD.

What the firmware does in each loop iteration:

1. Connect to the card and establish the secure channel.
2. Submit the PIN via CryptnoxWallet::verifyPin.
3. On success print "PIN accepted"; on failure halt immediately to protect the card's retry counter.

Warning

On PIN rejection the firmware enters an infinite halt. Each wrong PIN decrements the card's retry counter; reaching 0 permanently blocks the PIN and requires the PUK to unblock. Fix DEMO_↩ PIN before flashing again.

```
/*
 * SPDX-License-Identifier: LGPL-3.0-or-later
 * Copyright (c) 2026 Cryptnox SA
 */


#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "driver/spi_master.h"
#include "esp_log.h"
#include "esp_wifi.h"
#include "esp_netif.h"
#include "esp_event.h"
#include "nvs_flash.h"
#include "pn532.h"
#include "CryptnoxWallet.h"
#include "Pn532NfcTransport.h"
#include "ESP32Logger.h"
#include "esp32_crypto_provider.h"
#include "ESP32Platform.h"
#include "config.h"

/* - SPI wiring -- ESP32-S3 dev kit + Keyestudio PN532 breakout --- */
#define SPI_MOSI            11
#define SPI_MISO            13
#define SPI_SCLK            12
#define SPI_MAX_TRANSFER_SZ 4096
#define SPI_PIN_UNUSED      (-1)
#define NFC_CS              10

static const char *const TAG           = "verify_pin";
static const uint32_t    LOOP_DELAY_MS  = 1000U;
static const uint32_t    WIFI_TIMEOUT_MS = 10000U;
static const int         WIFI_MAX_RETRY  = 5;

#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT      BIT1

static EventGroupHandle_t s_wifi_event_group;
static int                s_retry_num = 0;

static void wifi_event_handler(void *arg, esp_event_base_t event_base,
                               int32_t event_id, void *event_data)
{
    (void)arg;
    (void)event_data;
    if ((event_base == WIFI_EVENT) && (event_id == WIFI_EVENT_STA_START)) {
        esp_wifi_connect();
    } else if ((event_base == WIFI_EVENT) &&
               (event_id == WIFI_EVENT_STA_DISCONNECTED)) {
        if (s_retry_num < WIFI_MAX_RETRY) {
            esp_wifi_connect();
            s_retry_num++;
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
    } else if ((event_base == IP_EVENT) && (event_id == IP_EVENT_STA_GOT_IP)) {
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    } else {
        /* other events ignored */
```

```
    }
}

static void wifi_start(void)
{
    s_wifi_event_group = xEventGroupCreate();
    s_retry_num = 0;
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    (void)esp_netif_create_default_wifi_sta();
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));
    esp_event_handler_instance_t h_any;
    esp_event_handler_instance_t h_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(
        WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler, NULL, &h_any));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(
        IP_EVENT, IP_EVENT_STA_GOT_IP, &wifi_event_handler, NULL, &h_ip));
    wifi_config_t wifi_cfg;
    (void)memset(&wifi_cfg, 0, sizeof(wifi_cfg));
    (void)strncpy((char *)wifi_cfg.sta.ssid,     WIFI_SSID,
                  sizeof(wifi_cfg.sta.ssid)     - 1U);
    (void)strncpy((char *)wifi_cfg.sta.password, WIFI_PASSWORD,
                  sizeof(wifi_cfg.sta.password) - 1U);
    wifi_cfg.sta.threshold.authmode = WIFI_AUTH_WPA2_PSK;
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_cfg));
    ESP_ERROR_CHECK(esp_wifi_start());
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
                                           WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
                                           pdFALSE, pdFALSE,
                                           pdMS_TO_TICKS(WIFI_TIMEOUT_MS));
    if ((bits & WIFI_CONNECTED_BIT) != 0U) {
        ESP_LOGI(TAG, "WiFi connected");
    } else {
        ESP_LOGW(TAG, "WiFi connect failed -- TRNG entropy may be reduced");
    }
    (void)esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP, h_ip);
    (void)esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID, h_any);
    vEventGroupDelete(s_wifi_event_group);
}

static void run_verify_pin_loop(CryptnoxWallet &wallet)
{
    /* Replace with the PIN set on the card (4-9 ASCII digits). */
    static const uint8_t DEMO_PIN[CW_MAX_PIN_LENGTH] = {
        '0', '0', '0', '0', '0', '0', '0', '0', '0'
    };

    bool keep_running = true;
    while (keep_running) {
        CW_SecureSession session{};
        bool connected = wallet.connect(session);

        if (!connected) {
            ESP_LOGW(TAG, "Card not detected");
        }

        if (connected) {
            bool pinOk = wallet.verifyPin(session,
                                          DEMO_PIN,
                                          static_cast<uint8_t>(CW_MAX_PIN_LENGTH));
            if (pinOk) {
                ESP_LOGI(TAG, "PIN accepted");
            } else {
                /* CRITICAL: do NOT retry -- each wrong PIN decrements the card's
                 * retry counter. Reaching 0 permanently blocks the PIN.
                 * Fix DEMO_PIN before flashing again. */
                ESP_LOGE(TAG, "PIN rejected -- halting to protect retry counter");
                keep_running = false;
            }
        }

        wallet.disconnect(session);

        if (keep_running) {
            vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
        }
    }

    while (true) {
        vTaskDelay(pdMS_TO_TICKS(LOOP_DELAY_MS));
    }
}

extern "C" void app_main(void)
{
```

```
    esp_err_t nvs_ret = nvs_flash_init();
    if ((nvs_ret == ESP_ERR_NVS_NO_FREE_PAGES) ||
        (nvs_ret == ESP_ERR_NVS_NEW_VERSION_FOUND)) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        nvs_ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(nvs_ret);
    wifi_start();

    spi_bus_config_t buscfg = {};
    buscfg.mosi_io_num     = SPI_MOSI;
    buscfg.miso_io_num     = SPI_MISO;
    buscfg.sclk_io_num     = SPI_SCLK;
    buscfg.quadwp_io_num   = SPI_PIN_UNUSED;
    buscfg.quadhd_io_num   = SPI_PIN_UNUSED;
    buscfg.max_transfer_sz = SPI_MAX_TRANSFER_SZ;
    ESP_ERROR_CHECK(spi_bus_initialize(SPI2_HOST, &buscfg, SPI_DMA_CH_AUTO));

    pn532_t nfc = {};
    pn532_config_t nfc_cfg = {};
    nfc_cfg.spi_host      = SPI2_HOST;
    nfc_cfg.pin_cs        = NFC_CS;
    nfc_cfg.skip_bus_init = true;
    esp_err_t nfc_ret = pn532_init(&nfc, &nfc_cfg);

    if (nfc_ret == ESP_OK) {
        ESP32Logger logger;
        (void)logger.begin(115200UL);

        ESP32CryptoProvider cryptoProvider;
        ESP32Platform platform;
        Pn532NfcTransport nfcTransport(&nfc, logger);
        CryptnoxWallet wallet(nfcTransport, logger, cryptoProvider, platform);

        if (wallet.begin()) {
            run_verify_pin_loop(wallet);
        } else {
            ESP_LOGE(TAG, "Wallet init failed");
        }
    } else {
        ESP_LOGE(TAG, "PN532 init failed");
    }
}
```