Cryptnox SDK for Arduino Manual

Release 1.0.0

June 20, 2026

Chapter 1

cryptnox-sdk-arduino

1.0.0.1 cryptnox-sdk-arduino

Arduino library for managing Cryptnox Hardware Wallet

`cryptnox-sdk-arduino` is an Arduino library that enables the use of the **Cryptnox Hardware Wallet** on Arduino UNO R4 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

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

Hardware Wallet	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
Arduino UNO R4 Minima	Renesas RA4M1 (Cortex-M4)	Recommended for compact projects
Arduino UNO R4 WiFi	Renesas RA4M1 + ESP32-S3	Adds Wi-Fi/Bluetooth via the on-board ESP32-S3

1.0.2 Installation

Important

The library cannot be installed from the Arduino Library Manager alone: it depends on patched versions of `AESLib`, `Adafruit_PN532`, and on Renesas-core memory optimisations without which sketches will either fail to compile or overflow flash on the UNO R4. The `setup.bat` installer below applies all of these automatically.

1.0.2.1 Prerequisites

1. Install **`Arduino IDE 2.x`**.
2. Add the **Arduino UNO R4** board support package: *Tools* → *Board* → *Boards Manager* → search for `Arduino UNO R4 Boards` → **Install**.
3. Install **`git`** and make sure it is in `PATH`.

1.0.2.2 Install (Windows — `setup.bat`)

```
git clone --recurse-submodules https://github.com/cryptnox/cryptnox-sdk-arduino.git
cd cryptnox-sdk-arduino
setup.bat
```

`setup.bat` performs every step the library needs to compile and run:

1. backs up the current `Arduino\libraries` folder (sibling directory, timestamped),
2. copies the SDK into `libraries\CryptnoxWallet`,
3. clones each pinned third-party dependency (`AESLib`, `Adafruit_BusIO`, `Adafruit_PN532`, `ArduinoHttpClient`, `Crypto`, `micro-ecc`, `trng`),
4. applies the patches from `patches\` — **required**, not optional:
   - `AESLib_renesas.patch` (compile on the Renesas core),
   - `AESLib_no_iostream.patch` (145 KB flash saved),
   - `Adafruit_PN532_timeout.patch` (longer card-detection window),
   - `Adafruit_PN532_extended_frame.patch` (extended-frame APDU support so certificate pages $>$ 255 bytes deliver intact),
5. runs `scripts\enable_memory_optimization.bat` to strip the AVR-compat float-printf hard pull and add `-fno-exceptions -fno-rtti` + uECC curve trimming to the core (11 KB more flash saved).

Pass `--reset` to wipe `Arduino\libraries` after the backup. Pass a path as the first argument to target a non-default libraries directory.
When the script finishes, **restart the Arduino IDE**, then open *File* → *Examples* → **CryptnoxWallet** → **Connect**, select your board (*Tools* → *Board* → **Arduino UNO R4 Minima** or **WiFi**), pick the serial port, and **Upload**.

Remarks

macOS / Linux: no installer is shipped yet. Follow DEVELOPMENT.md to clone the repo into your `Arduino/libraries` directory and apply the patches under `patches\` by hand — they are unified diffs and `patch -p1` works on all four.

See DEVELOPMENT.md for the source-build workflow, coding style, and release process.

1.0.3 Hardware setup

Attention

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

Wiring shown for the **Arduino UNO R4 WiFi** (identical pin numbering on the Minima variant).

1.0.3.1 Arduino UNO R4 and PN532 NFC — SPI interface

PN532 Pin	Arduino Pin	Wire Color
VCC	3.3V	Red
GND	GND	Black
SCK	D13	Blue
MISO	D12	Green
MOSI	D11	Yellow
SS	D10	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 Arduino UNO R4 and PN532 NFC — I²C interface

PN532 Pin	Arduino Pin	Wire Color
VCC	3.3V	Red
GND	GND	Black
SDA	A4	Yellow
SCL	A5	Blue

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

Full sketches live under `examples/`. Each one is also visible from the Arduino IDE under *File* → *Examples* → *CryptnoxWallet*. The snippets below show the interesting part — see the linked `.ino` for the complete file (setup, error handling, comments).

1.0.4.1 1. Connect & read card info — `Connect.ino`

```
#include <CryptnoxWallet.h>
#include <SPI.h>

ArduinoLoggerAdapter   serialAdapter;
PN532Adapter           nfc(serialAdapter, /*SS=*/10U, &SPI);
ArduinoCryptoProvider  cryptoProvider;
ArduinoPlatform        platform;
CryptnoxWallet         wallet(nfc, serialAdapter, cryptoProvider, platform);

void setup() {
    serialAdapter.begin(115200);
    SPI.begin();
    if (!wallet.begin()) {
        serialAdapter.println(F("PN532 init failed"));
        while (1);
```

```
    }
}

void loop() {
    CW_SecureSession session;
    if (wallet.connect(session)) {
        serialAdapter.println(F("Card connected, secure channel established"));
        CW_CardInfo info;
        if (wallet.getCardInfo(session, &info)) {
            serialAdapter.print(F("Owner name : "));
            serialAdapter.println(info.name);
            serialAdapter.print(F("Owner email: "));
            serialAdapter.println(info.email);
        } else {
            serialAdapter.println(F("getCardInfo failed (channel error or parse error)"));
        }
    } else {
        serialAdapter.println(F("Card not detected or secure channel failed"));
    }
    wallet.disconnect(session);
    delay(1000);
}
```

1.0.4.2 2. Verify the PIN — `VerifyPin.ino`

```
const char* pin = "000000000";  // must match the PIN set on the card

if (!wallet.verifyPin(session,
                      reinterpret_cast<const uint8_t*>(pin),
                      (uint8_t)strlen(pin))) {
    serialAdapter.println(F("Wrong PIN -- halting to protect retry counter"));
    wallet.disconnect(session);
    while (1);   // each wrong attempt burns one retry; do NOT loop
}
serialAdapter.println(F("PIN accepted"));
```

1.0.4.3 3. Sign a 32-byte hash — `Sign.ino`

```
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);
CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
                      reinterpret_cast<const uint8_t*>("000000000"), 9U);

CW_SignResult sig = wallet.sign(req);
if (sig.errorCode == CW_OK) {
    // sig.signature = r[32] || s[32]   (raw, ready to forward / DER-encode)
}
CW_Utils::secure_wipe(hash, sizeof(hash));
CW_Utils::secure_wipe(sig.signature, sizeof(sig.signature));
```

For a full end-to-end real-world flow (WiFi + JSON-RPC + EIP-1559 tx), see `examples/Usdc_Signing/` which signs and broadcasts a USDC transfer on Sepolia.

Note

The card must have a seed loaded before signing will work. The easiest way to provision it is with the `Cryptnox CLI`:

```
cryptnox init
cryptnox seed generate
```

Advanced users can script the same flow via `cryptnox-sdk-py` (`card.generate_seed(pin)` or `card.load_seed(seed, pin)`). The default PIN shown above (`000000000`) must match the one set during initialization.

Warning

The PIN `"000000000"` is a demo placeholder. Set a strong PIN when initialising the card, change any factory default before storing real funds, and never commit source files containing a real PIN.

1.0.5 Troubleshooting

- **The library doesn't appear in the Arduino IDE** → restart the IDE after installation.
- **The PN532 is not detected** → check the wiring and the switch configuration (SPI vs I²C). They must match the interface defined in your sketch (`USE_SPI` or `USE_I2C`).
- **Upload fails** → verify that the correct board (Arduino UNO R4 Minima / WiFi) and the correct port are selected in the *Tools* menu.
- **`Sign failed` with a non-zero error code** → the card may not have a seed loaded. Provision it once with the `Cryptnox CLI`.

1.0.6 Documentation

The generated documentation for this project is available `here`.

1.0.7 Contributing

Contributions are welcome! See CONTRIBUTING.md for how to propose changes, and DEVELOPMENT.md for the development setup.

1.0.8 License

`cryptnox-sdk-arduino` 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 Public API

Types and classes application code interacts with directly.

Classes

- struct CW_CardInfo

  Subset of the Cryptnox card info returned by APDU 0x80FA0000.
- struct CW_SignRequest

  Request parameters for CryptnoxWallet::sign.
- class CryptnoxWallet

  High-level interface for interacting with a Cryptnox Hardware Wallet over NFC.
- struct CW_SecureSession

  Holds cryptographic session state for reentrant secure channel operations.

2.1.1 Detailed Description

Types and classes application code interacts with directly.
Includes CryptnoxWallet (the main entry point), the sign request / result structs, CW_CardInfo, and CW_SecureSession.

2.2 Secure channel protocol

Low-level secure messaging implementation.

Classes

- class CW_SecureChannel

  Implements the Cryptnox secure channel protocol over NFC.

2.2.1 Detailed Description

Low-level secure messaging implementation.
CW_SecureChannel is composed inside CryptnoxWallet and is not normally used directly. Documented here for callers that need to drive the activation sequence manually (e.g. custom retry policies, fuzzing).

2.3 Adapter interfaces

Abstract contracts a host integration must implement.

Classes

- class CW_CryptoProvider

  Abstract interface for cryptographic operations used by CW_SecureChannel.
- class CW_Logger

  Abstract interface for serial/debug output.
- class CW_NfcTransport

  Abstract interface for NFC transport operations.
- class CW_Platform

  Abstract interface for platform-specific operations used by the SDK.

2.3.1 Detailed Description

Abstract contracts a host integration must implement.
Provide concrete implementations of CW_NfcTransport, CW_CryptoProvider, CW_Logger, and CW_Platform when porting the SDK to a new MCU or operating system.

2.4 Utilities & shared definitions

Platform-independent helpers and shared constants.

Classes

- class CW_Utils

  Portable utility functions for cryptographic and security operations.

Enumerations

- enum CW_Curve { CW_CURVE_SECP256R1 = 0 , CW_CURVE_SECP256K1 = 1 }

  Portable curve identifier used throughout the SDK.

2.4.1 Detailed Description

Platform-independent helpers and shared constants.
Includes CW_Utils (constant-time compare, secure wipe, safe memcpy, RNG), the `CW_CERT_*` / `CW_SIGN_*` error codes, and the trusted-CA key table (CW_TrustedKeys.h).

2.4.2 Enumeration Type Documentation

2.4.2.1 CW_Curve

`enum CW_Curve`
Portable curve identifier used throughout the SDK.
Replaces direct references to uECC_Curve_t at every API boundary so the abstract interfaces (CW_CryptoProvider, CW_SecureChannel) remain decoupled from any specific ECC back-end.

Enumerator

CW_CURVE_SECP256R1	NIST P-256 / secp256r1
CW_CURVE_SECP256K1	Koblitz secp256k1

Definition at line 151 of file CW_Defs.h.

2.5 Arduino concrete adapters

Concrete `CW_NfcTransport` / `CW_CryptoProvider` / `CW_Logger` / `CW_Platform` implementations for the Arduino UNO R4 (RA4M1).

Classes

- class ArduinoCryptoProvider

  `CW_CryptoProvider` implementation for the Arduino UNO R4 (RA4M1).
- class ArduinoLoggerAdapter

  `CW_Logger` implementation wrapping Arduino's `HardwareSerial`.
- class ArduinoPlatform

  `CW_Platform` implementation using Arduino's blocking `delay()`.
- class NullLoggerAdapter

  No-op `CW_Logger` — guarantees nothing reaches the serial port.
- class PN532Adapter

  `CW_NfcTransport` implementation over the Adafruit_PN532 driver.

Enumerations

- enum class PN532Interface { PN532Interface::SPI_HARDWARE , PN532Interface::SPI_SOFTWARE , PN532Interface::I2C , PN532Interface::UART }

  Physical wiring used to reach the PN532 reader.

2.5.1 Detailed Description

Concrete `CW_NfcTransport` / `CW_CryptoProvider` / `CW_Logger` / `CW_Platform` implementations for the Arduino UNO R4 (RA4M1).
Each class in this group implements one of the abstract interfaces declared in Adapter interfaces using libraries available through the Arduino Library Manager:

Adapter	Interface	Backing library / hardware
PN532Adapter	`CW_NfcTransport`	Adafruit_PN532 (SPI / I2C / UART)
ArduinoCryptoProvider	`CW_CryptoProvider`	AESLib + SHA512/SHA256 + micro-ecc + RA4M1 TRNG
ArduinoLoggerAdapter	`CW_Logger`	Arduino `HardwareSerial` (development builds)
NullLoggerAdapter	`CW_Logger`	No-op (production builds — keeps APDU/key material off the UART)
ArduinoPlatform	`CW_Platform`	Arduino blocking `delay()`

2.5.2 Enumeration Type Documentation

2.5.2.1 PN532Interface

`enum class PN532Interface [strong]`
Physical wiring used to reach the PN532 reader.
Set internally by the constructor matching the wiring the caller used. Exposed for diagnostics — application code does not normally need to read it.

Mode	Pros	Cons
`SPI_HARDWARE`	Fastest, uses the MCU SPI peripheral.	Pins are fixed by the SPI block.
`SPI_SOFTWARE`	Any GPIO; useful when SPI is in use.	Slower, more CPU.
`I2C`	2 wires + IRQ + RESET.	Slower than SPI; needs pull-ups.
`UART`	2 wires + RESET; long traces tolerated.	Slowest; consumes a HardwareSerial.

Enumerator

SPI_HARDWARE	Hardware SPI via an `SPIClass` instance.
SPI_SOFTWARE	Bit-banged SPI on four arbitrary GPIOs.
I2C	I2C via a `TwoWire` instance.
UART	UART via a `HardwareSerial` instance.

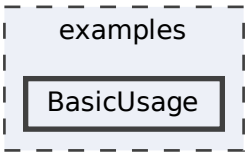
Definition at line 47 of file PN532Adapter.h.

Chapter 3

Directory Documentation

3.1 examples/BasicUsage Directory Reference

Directory dependency graph for BasicUsage:



Files

- file BasicUsage.ino

3.1.1 Detailed Description

3.1.1.0.1 cryptnox-sdk-arduino

Arduino library for managing Cryptnox Hardware Wallet

3.1.2 BasicUsage — End-to-End Walkthrough (SPI or I^{2}C)

A single self-contained sketch that exercises the **full Cryptnox flow** on Arduino: pick SPI or I^{2}C 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 sketch 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	Arduino UNO R4 Minima or WiFi
SDK	Installed via `setup.bat` — see `installation`

3.1.2.2 Quick start

1. **Pick the bus** at the top of `BasicUsage.ino` — uncomment exactly one:
   ```
   #define USE_SPI      // SCK/MISO/MOSI on D13/12/11, SS on D10
   // #define USE_I2C   // SDA on A4, SCL on A5
   ```
2. **Set the PIN** to match `cryptnox init`:
   ```
   #define DEFAULT_PIN  "000000000"
   ```
3. *File* → *Examples* → *CryptnoxWallet* → *BasicUsage*, select the board and serial port.
4. **Upload**, open the **Serial Monitor** at *115200 baud*, place the card on the PN532 antenna.

3.1.2.2.1 Expected output

```
Card connected and secure channel established
Signing test hash...
Signature received (64 bytes raw r||s)
  R[0..7]: 7C 1F 3A 92 5E 0B 8C D4
  S[0..7]: 12 E0 BC 4F A7 88 09 67
Card processed successfully
```

Note

The SDK uses `inDataExchange16` internally to handle PN532 extended frames, so manufacturer-certificate pages up to ∼411 bytes deliver correctly on both SPI and I²C.

3.1.2.3 How it works

```
setup():
  Serial / SPI(or Wire) bring-up
  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
  delay(1000)
```

3.1.2.4 Step-by-step code

1. Interface selection — exactly one of:

```
#define USE_SPI
// #define USE_I2C
```

On `USE_SPI` the sketch instantiates:

```
PN532Adapter nfc(serialAdapter, /*SS=*/10U, &SPI);
```

On `USE_I2C` it wires the optional IRQ/RST pins (D2/D3). These are typically **not connected** on the Keyestudio Ks0259 shield, and the upstream Adafruit library polls the IRQ pin in software regardless, so the sketch works with these pins floating on that shield:

```
PN532Adapter nfc(serialAdapter, /*IRQ=*/2U, /*RST=*/3U, &Wire);
```

2. Bring up the bus and reader:

```
void setup() {
    serialAdapter.begin(115200);
    delay(1000);                          // UNO R4: wait for USB-CDC Serial
#if defined(USE_SPI)
    SPI.begin();
#elif defined(USE_I2C)
    Wire.begin();
#endif
    if (!wallet.begin()) {
        serialAdapter.println(F("PN532 init failed"));
        while (1) {}
    }
}
```

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);
    CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
                          reinterpret_cast<const uint8_t*>(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 sketch is a demo. Before shipping firmware to end-users:

- **Silence Serial.** Replace `ArduinoLoggerAdapter serialAdapter;` with `NullLoggerAdapter serialAdapter;` and drop the `serialAdapter.begin()` call. The null adapter is a no-op (zero code-size / runtime cost) and stops the SDK from emitting APDU and key material on USB-CDC.
- **Move the PIN off flash.** A hardcoded `DEFAULT_PIN` lives in `.rodata` and is recoverable via SWD/JTAG. In production read the PIN at runtime (keypad, BLE prompt, companion chip) 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 define USE_SPI or USE_I2C...` (build error)	Neither macro defined	Uncomment exactly one of `#define USE_SPI` / `#define USE_I2C`
`PN532 init failed`	Reader wiring / bus mode switches	Check VCC = 3.3 V, the switches and the configured pins — see `hardware setup`

Symptom	Cause	Fix
`Sign failed, error↩ Code: 0x81`	No seed on the card	`cryptnox seed generate`
`Sign failed, error↩ Code: 0x82`	Wrong PIN	Edit `DEFAULT_PIN`, re-upload — see VerifyPin
Sketch overflows flash	Memory-optimisation script not run	Re-run `setup.bat` — it calls `scripts/enable_memory↩ _optimization.bat`

3.1.2.7 License

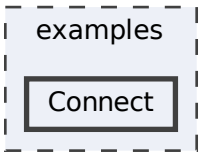
`cryptnox-sdk-arduino` is dual-licensed:

- **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 examples/Connect Directory Reference

Directory dependency graph for Connect:



Files

- file Connect.ino

3.2.1 Detailed Description

3.2.1.0.1 cryptnox-sdk-arduino

Arduino library for managing Cryptnox Hardware Wallet

3.2.2 Connect — Secure Channel + Card Info

Open the Cryptnox Hardware Wallet secure channel from an Arduino UNO R4 and read back the card owner's **name** and **email**. This is the **safest starting point**: the sketch never sends a PIN, so it cannot lock the card.

3.2.2.1 Requirements

Component	Details
Hardware Wallet	Cryptnox Hardware Wallet, initialised (`cryptnox init`)
NFC reader	PN532 over SPI — see `hardware setup`
Board	Arduino UNO R4 Minima or WiFi
SDK	Installed via `setup.bat` — see `installation`

No PIN, no seed required.

3.2.2.2 Quick start

1. *File* → *Examples* → *CryptnoxWallet* → *Connect*
2. *Tools* → *Board* → *Arduino UNO R4 Minima* (or *WiFi*) and *Tools* → *Port*
3. **Upload**, open the **Serial Monitor** at *115200 baud*
4. Place the card on the PN532 antenna

3.2.2.2.1 Expected output

```
PN532 information
 +- Raw firmware: 0x32010607
 +- IC Chip: PN532
 +- Firmware: 1.6
 +- Features: MIFARE + ISO-DEP + FeliCa (0x7)
Card connected, secure channel established
Owner name : Alice
Owner email: alice@cryptnox.com
```

3.2.2.3 How it works

```
wallet.begin()                Bring up SPI + reset the PN532
       |
wallet.connect(session)       SELECT + cert chain verify (secp256r1)
       |                      + ECDH key agreement + mutual auth
       |                      → session.aesKey / macKey / iv populated
       |
wallet.getCardInfo(session)   Secured APDU (AES-CBC + MAC)
       |                      Decrypt response, parse name & email
       |
wallet.disconnect(session)    Zero session keys
```

3.2.2.4 Step-by-step code

Wire the adapters together (declared once at file scope):

```
ArduinoLoggerAdapter   serialAdapter;
PN532Adapter           nfc(serialAdapter, /*SS=*/10U, &SPI);
ArduinoCryptoProvider  cryptoProvider;
ArduinoPlatform        platform;
CryptnoxWallet         wallet(nfc, serialAdapter, cryptoProvider, platform);
```

Bring up the reader in `setup()`:

```
serialAdapter.begin(115200);
delay(1000);                          // UNO R4: wait for USB-CDC Serial
SPI.begin();
if (!wallet.begin()) {
    serialAdapter.println(F("PN532 init failed"));
    while (1) {}
}
nfc.printFirmwareVersion();
```

Open the channel and read the card in `loop()`:

```
CW_SecureSession session;
if (wallet.connect(session)) {
    serialAdapter.println(F("Card connected, secure channel established"));
    CW_CardInfo info;
    if (wallet.getCardInfo(session, &info)) {
        serialAdapter.print(F("Owner name : "));
        serialAdapter.println(info.name);
```

```
        serialAdapter.print(F("Owner email: "));
        serialAdapter.println(info.email);
    } else {
        serialAdapter.println(F("getCardInfo failed (channel error or parse error)"));
    }
} else {
    serialAdapter.println(F("Card not detected or secure channel failed"));
}
wallet.disconnect(session);
```

3.2.2.5 Troubleshooting

Symptom	Cause	Fix
`PN532 init failed`	Reader wiring / power / mode switches	Check VCC = 3.3 V, the SCK/MISO/MOSI/SS pinout, and SPI mode switches (`SW0=HIGH, SW1=LOW`) — see `hardware setup`
`Card not detected or secure channel failed`	Card not on 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
`APDU exchange failed!`	NFC link lost mid-exchange	Hold the card steady while the PN532 LED is on

3.2.2.6 License

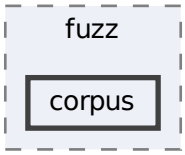
`cryptnox-sdk-arduino` is dual-licensed:

- **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`

3.3 src/cryptnox-sdk-cpp/fuzz/corpus Directory Reference

Directory dependency graph for corpus:



3.3.1 Detailed Description

3.3.2 Fuzz corpus — DER parser seeds

Place binary seed files in this directory before running `fuzz_der`.

3.3.3 Recommended seeds

3.3.3.1 DER ECDSA signatures (`selector = 0x00`)

Files must start with `0x00`, followed by a raw DER ECDSA signature. A minimal valid P-256 DER signature (70–72 bytes) looks like: `30 44 02 20 <r[32]> 02 20 <s[32]>`
Extract a real signature from any Cryptnox transaction and prepend `0x00`.

3.3.3.2 Manufacturer certificate blobs (`selector = 0x01`)

Files must start with `0x01`, followed by a raw DER X.509 certificate. Obtain a real Cryptnox manufacturer certificate via `CW_SecureChannel::getManufacturerCertificate()` and prepend `0x01`.

3.3.4 Example (Linux/macOS)

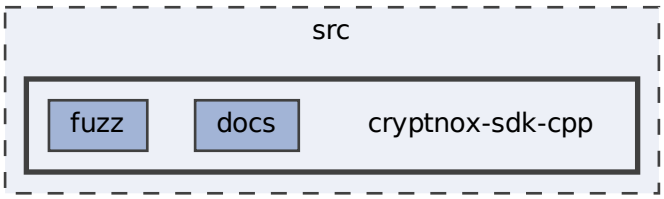
```
# Prepend selector byte 0x01 to an existing DER cert
printf '\x01' | cat - mfr_cert.der > corpus/cert_01.bin

# Prepend selector byte 0x00 to an existing DER signature
printf '\x00' | cat - sig.der > corpus/sig_00.bin
```

3.3.5 Running the fuzzer

```
cd cryptnox-sdk-cpp/fuzz
mkdir build && cd build
cmake .. -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++
make
./fuzz_der ../corpus/ -max_len=512 -jobs=4
```

3.4 src/cryptnox-sdk-cpp Directory Reference

Directory dependency graph for cryptnox-sdk-cpp:



Directories

- directory docs
- directory fuzz

Files

- file CryptnoxWallet.cpp
  Implementation of the high-level CryptnoxWallet API.
- file CryptnoxWallet.h
  High-level API for interacting with a Cryptnox Hardware Wallet over NFC.
- file CW_CryptoProvider.h
  Abstract cryptographic primitives interface.
- file CW_Defs.h
  Shared constants, error codes, and session state for the SDK.
- file CW_Logger.h
  Abstract logging interface.
- file CW_NfcTransport.h
  Abstract NFC transport interface.
- file CW_Platform.h
  Abstract platform interface for timing primitives.
- file CW_SecureChannel.cpp
  Implementation of the Cryptnox secure channel protocol.
- file CW_SecureChannel.h
  Cryptnox secure channel protocol over NFC.
- file CW_TrustedKeys.h
  Cryptnox CA public keys used for offline certificate verification.
- file CW_Utils.cpp
  Implementation of the platform-independent utility helpers.
- file CW_Utils.h
  Platform-independent security and memory utilities.
- file platform_compat.h
  Arduino compatibility shims for non-Arduino (plain C++) builds.

3.4.1 Detailed Description

3.4.1.0.1 cryptnox-sdk-cpp

Platform-independent C++ core SDK for Cryptnox Hardware Wallet
Download this documentation as PDF

`cryptnox-sdk-cpp` is the **shared C++ core SDK** for the **Cryptnox Hardware Wallet**. It implements the card-side protocol — secure channel establishment (SELECT → certificate → ECDH → mutual auth), APDU framing, PIN verification, signing, and user-data writing — independently of any target platform, NFC reader, or crypto library.

Important

> **This SDK is not usable on its own.** It exposes three abstract interfaces (`CW_NfcTransport`, `CW_CryptoProvider`, `CW_Logger`) that **must be implemented by a host integration**. It ships no transport driver, no crypto backend, and no logging output.

3.4.1.1 Used by

This core is consumed as a submodule by the platform-specific SDKs:

Integration	Repository
ESP32-S3 (ESP-IDF v5.5)	cryptnox/cryptnox-sdk-esp32
Arduino R4 (Renesas RA4M1)	cryptnox/cryptnox-sdk-arduino

If you want to talk to a Cryptnox card on real hardware, **start from one of those repositories**.

3.4.1.2 Porting to a new platform

This repository is intended as the **starting point for porting the SDK to a new platform** (another MCU family, a desktop OS, a different NFC reader, a different crypto backend, etc.).
A port consists of providing concrete implementations of the three adapter interfaces:

Interface	What you must provide
CW_NfcTransport	Driver for your NFC reader (PN532, PN7150, PC/SC, ...)
CW_CryptoProvider	SHA-256/512, AES-CBC, ECDH, EC key generation, RNG (mbedTLS, BearSSL, OpenSSL, hardware peripheral, ...)
CW_Logger	Output sink (UART, stdout, syslog, network, ...)

Then drop this repository in as a submodule (or copy of its sources) inside your project, build it together with your adapters, and instantiate CryptnoxWallet with the three injected dependencies. The existing platform SDKs (cryptnox-sdk-esp32, cryptnox-sdk-arduino) are useful references for a complete port.

3.4.1.3 What's inside

File	Role
CryptnoxWallet.{h,cpp}	High-level API: begin, connect, verifyPin, sign, writeUserData, disconnect
CW_SecureChannel.{h,cpp}	Secure channel protocol (mutual auth, session keys, encrypted APDU exchange)
CW_NfcTransport.h	**Adapter interface** — NFC reader contract
CW_CryptoProvider.h	**Adapter interface** — SHA-256/512, AES-CBC, ECDH, EC keygen, RNG
CW_Logger.h	**Adapter interface** — debug/serial output
CW_TrustedKeys.h	Cryptnox CA public keys used to verify card certificates
CW_Defs.h, CW_Utils.{h,cpp}	Constants, error codes, small helpers
platform_compat.h	Shim for non-Arduino targets

The three adapter interfaces are the only contract a host must satisfy. Everything else is self-contained.

3.4.1.4 Integrating the core

A host integration injects its three adapters into CryptnoxWallet:

```
#include "CryptnoxWallet.h"

// MyXxx = concrete adapters provided by the platform SDK:
//   - MyNfcTransport   : public CW_NfcTransport
//   - MyCryptoProvider : public CW_CryptoProvider
//   - MyLogger         : public CW_Logger

MyLogger          logger;
```

```
MyCryptoProvider  crypto;
MyNfcTransport    transport(/* platform-specific args */);
CryptnoxWallet    wallet(transport, logger, crypto);

if (!wallet.begin()) { /* reader init failed */ }

CW_SecureSession session;
if (wallet.connect(session)) {
    // wallet.verifyPin(...), wallet.sign(...), wallet.writeUserData(...)
    wallet.disconnect(session);
}
```

Full runnable examples (PIN verify, transaction signing, full ESP-IDF / Arduino boilerplate) live in the platform SDKs linked above.

3.4.1.5 Building standalone

There is **no standalone build target** in this repository. The CI workflow runs `cppcheck` static analysis only (`.github/workflows/static_analysis.yml`); host-supplied headers (`uECC.h`, `mbedtls/*`, platform logger) are not vendored, so `missingInclude` is suppressed.
To compile and exercise the code, use one of the platform SDKs.

3.4.1.6 Documentation

The generated documentation for this project is available `here`.

3.4.1.7 License

`cryptnox-sdk-cpp` 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

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

3.5 src/cryptnox-sdk-cpp/docs Directory Reference

Directory dependency graph for docs:

Files

- file patch_latex.py

3.6 examples Directory Reference

Directories

- directory BasicUsage

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

3.6.1 Detailed Description

3.6.1.0.1 cryptnox-sdk-arduino

Arduino library for managing Cryptnox Hardware Wallet

3.6.2 Examples

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

3.6.2.1 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	`Arduino UNO R4 Minima` or `Arduino UNO R4 WiFi` (the **WiFi** variant is required for UsdcSigning)
IDE	`Arduino IDE 2.x` with the **Arduino UNO R4 Boards** core installed
SDK	Installed via `setup.bat` from the repository root — see `installation`

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)
```

3.6.2.2 Available examples

Example	What it does
Connect	Opens the secure channel and reads back the card owner's name & email. **Safest first sketch** — 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 sketch: pick SPI **or** I²C, open the channel, sign a hash. Good reference for production wiring.
UsdcSigning	Real-world flow on UNO R4 WiFi: build an EIP-1559 USDC transfer, sign it on the card, broadcast it on Sepolia.

3.6.2.3 How to run an example

1. **Install the library** by running `setup.bat` from the repository root (Windows). The script clones every pinned third-party dependency, applies the required patches, and runs the

memory-optimisation pass — without it the sketches may fail to compile or overflow flash. See installation.

2. **Restart the Arduino IDE.**
3. **Open the sketch** via *File* → *Examples* → *CryptnoxWallet* → <*example*>.
4. **Select the board**: *Tools* → *Board* → *Arduino UNO R4 Minima* (or *WiFi* — required for Usdc↩ Signing).
5. **Select the serial port**: *Tools* → *Port* → *...*
6. **(UsdcSigning only)** copy config.template.h to config.h and fill in your Wi-Fi credentials, RPC endpoint, PIN, addresses and amount. config.h is gitignored — do not commit it.
7. **Compile and upload**, then open the **Serial Monitor** at *115200 baud*. Place the card on the PN532 antenna when prompted.

Note

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

3.6.2.4 Adding a new example

Follow the conventions used by the existing sketches:

- Place each sketch in its own subdirectory under examples/, named in **PascalCase** (the Arduino IDE requires the .ino file to share the directory name).
- Start every source file with the SPDX + copyright header used by the rest of the repository.
- Add Doxygen tags (@file, @example, @brief) to the .ino so the sketch surfaces correctly on the generated docs site.
- If the sketch needs external secrets (Wi-Fi credentials, API 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 sketch and register it in the Available examples table above.

3.6.2.5 License

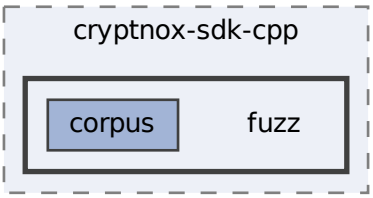
cryptnox-sdk-arduino is dual-licensed:

- **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

3.7 src/cryptnox-sdk-cpp/fuzz Directory Reference

Directory dependency graph for fuzz:



Directories

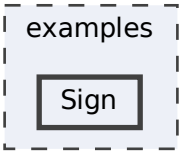
- directory corpus

Files

- file fuzz_der.cpp

3.8 examples/Sign Directory Reference

Directory dependency graph for Sign:



Files

- file Sign.ino

3.8.1 Detailed Description

3.8.1.0.1 cryptnox-sdk-arduino

Arduino library for managing Cryptnox Hardware Wallet

3.8.2 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 Arduino 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 sketch halts on `CW_SIGN_PIN_INCORRECT` (`0x82`) — do not remove the `while (1)` guard, and verify `DEMO_PIN` before uploading.

3.8.2.1 Requirements

Component	Details
Hardware Wallet	Cryptnox Hardware Wallet, initialised **and** seeded
NFC reader	PN532 over SPI — see `hardware setup`
Board	Arduino UNO R4 Minima or WiFi
SDK	Installed via `setup.bat` — see `installation`

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`).

3.8.2.2 Quick start

1. **Edit `DEMO_PIN`** in `Sign.ino` to match your card's PIN.
2. *File* → *Examples* → *CryptnoxWallet* → *Sign*, select the board + port.
3. **Upload**, open the **Serial Monitor** at *115200 baud*, place the card on the antenna.

3.8.2.2.1 Expected output

```
Signature OK
  r = 7C1F3A925E0B8CD4920FAB7C53E1B9D81F2A4C76E9D805BC5D2EAD12C3FA47CB
  s = 12E0BC4FA7880967DA1F0EBD83C2B791580AC4D62E16039F7B4CDDF13E2A89C5
  errorCode = 0x0
```

`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 (Ethereum, modern Bitcoin clients, ...).

3.8.2.3 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
       |                       over the secure channel.
       |
secure_wipe(hash, signature)   Zero local copies
       |
wallet.disconnect(session)     Zero session keys
```

3.8.2.3.1 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).

3.8.2.3.2 PIN: in payload vs pre-verified

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

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

3.8.2.4 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);
CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
                      reinterpret_cast<const uint8_t*>(DEMO_PIN),
                      strlen(DEMO_PIN));
```

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);
    while (1) {}                          // protect the retry counter
}
```

Wipe secrets before exiting the scope:

```
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.

3.8.2.5 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

3.8.2.6 Troubleshooting

Symptom	Cause	Fix
`Sign failed: 0x81`	No seed on card	`cryptnox seed generate`
`Sign failed: 0x82`	Wrong PIN	Edit `DEMO_PIN`, re-upload — 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

3.8.2.7 License

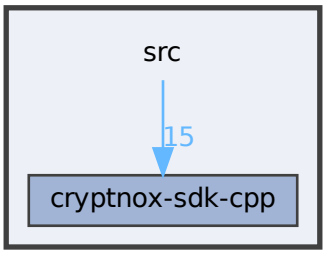
`cryptnox-sdk-arduino` is dual-licensed:

- **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 src Directory Reference

Directory dependency graph for src:



Directories

- directory cryptnox-sdk-cpp

Files

- file ArduinoCryptoProvider.cpp

  Implementation of the Arduino UNO R4 concrete crypto provider.
- file ArduinoCryptoProvider.h

  Concrete `CW_CryptoProvider` for the Arduino UNO R4 (RA4M1).
- file ArduinoLoggerAdapter.cpp

  Implementation of the HardwareSerial-backed logger.
- file ArduinoLoggerAdapter.h

Concrete `CW_Logger` over `HardwareSerial` for development builds.

- file ArduinoPlatform.cpp

  Implementation of the Arduino blocking-delay platform adapter.
- file ArduinoPlatform.h

  Concrete `CW_Platform` over Arduino's `delay()`.
- file CryptnoxWallet.h

  Umbrella include and module-group anchor for the CryptnoxWallet Arduino library.
- file NullLoggerAdapter.h

  Silent `CW_Logger` for production firmware.
- file PN532Adapter.cpp

  Implementation of the PN532 transport adapter.
- file PN532Adapter.h

  Concrete `CW_NfcTransport` over Adafruit_PN532 (SPI / I2C / UART).

3.10 examples/UsdcSigning Directory Reference

Directory dependency graph for UsdcSigning:

Files

- file config.template.h
- file keccak256.cpp

  Keccak-256 (SHA3 variant) hash implementation for Ethereum.
- file keccak256.h

  Keccak-256 (SHA3 variant) hash function for Ethereum.
- file UsdcSigning.ino
- file util.cpp
- file util.h

3.10.1 Detailed Description

3.10.1.0.1 cryptnox-sdk-arduino

Arduino library for managing Cryptnox Hardware Wallet

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

End-to-end demonstration of the Cryptnox Hardware Wallet on an Arduino UNO R4 WiFi: 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 Arduino 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 sketch halts on `CW_SIGN_PIN_INCORRECT` — verify `CARD_PIN` in `config.h` matches the value used during `cryptnox init` **before** uploading.

3.10.2.1 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 — see `hardware setup`
Board	**Arduino UNO R4 WiFi** (the Minima has no Wi-Fi)
RPC endpoint	A Sepolia JSON-RPC endpoint — `PublicNode` (no signup, default) or `Infura`
SDK	Installed via `setup.bat` — see `installation`

3.10.2.2 Quick start

1. **Create `config.h`** by copying the template:
   ```
   cp config.template.h 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

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

1. *File → Examples → CryptnoxWallet → UsdcSigning*
2. *Tools → Board → ∗∗Arduino UNO R4 WiFi∗∗∗ (not Minima) and ∗Tools → Port*
3. **Upload**, open the **Serial Monitor** at *115200 baud*, place the card on the PN532 antenna when prompted.

3.10.2.2.1 Expected output

```
PN532 OK
Connecting to WiFi....
fetchNonce: connecting to ethereum-sepolia-rpc.publicnode.com/
fetchNonce: status=200 nonce=0x5
[HASH]: 0x6679a2cd3064046397addbb97004b606df9281f624409fd36d2d24832db59c29
Place Cryptnox card on PN532 reader...
Card connected, secure channel established.
[SIG r]: 7C1F3A925E0B8CD4920FAB7C53E1B9D81F2A4C76E9D805BC5D2EAD12C3FA47CB
[SIG s]: 12E0BC4FA7880967DA1F0EBD83C2B791580AC4D62E16039F7B4CDDF13E2A89C5
yParity: 1
Sending...
[RPC] HTTP 200
[tx] hash=0xab12cd34ef56...
Transaction sent successfully!
```

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

3.10.2.3 How it works

```
 1. WiFi.begin(WIFI_SSID, WIFI_PASSWORD)
        |
 2. fetchNonce()                      eth_getTransactionCount over
        |                             TLS-pinned WiFiSSLClient
        |
 3. Build Tx2 struct                  chainId=11155111, nonce, fees,
        |                             gasLimit, to, value=0, data
        |
 4. encodeERC20Transfer()             0xa9059cbb || pad(to,32) ||
        |                             pad(AMOUNT_USDC,32)
        |
 5. rlpEncodeUnsignedTx()             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. determineYParity(hash, r, s)      Call the ecrecover precompile
        |                             with v=27, then v=28 -- keep the
        |                             one that returns ADDR_FROM
        |
11. rlpEncodeSignedTx()               Same RLP plus the signature fields
        |
12. sendRawTx()                       eth_sendRawTransaction → extract
        |                             "result":"0x..." tx hash
        |
13. secure_wipe(req.pin, ...)         Zero the PIN buffer
```

3.10.2.3.1 TLS pinning

`WiFiSSLClient` is initialised with `setCACert(WIFI_CA_CERT)`. The sketch ships with **ISRG Root X1** (Let's Encrypt — *Internet Security Research Group*) as a placeholder default. That root does **not** match PublicNode's current chain (`ethereum-sepolia-rpc.publicnode.com` is issued by Google Trust Services R4) nor Infura's (DigiCert), so for any real endpoint you will need to override `WIFI_CA_CERT` in `config.h`. Retrieve the correct root with:

```
openssl s_client -showcerts -servername HOST -connect HOST:443 </dev/null
```

For development only you can set `WIFI_DISABLE_CA_PINNING` in `config.h` to skip validation. The sketch prints a loud warning at boot when this is set.

Attention

> Never ship firmware with `WIFI_DISABLE_CA_PINNING` defined — the connection becomes trivially MITM-able and an attacker can feed the device crafted nonces / fees and have it sign whatever they want.

3.10.2.3.2 `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 sketch calls the `ecrecover precompile` at address `0x01` with `v = 27` (yParity = 0). If the recovered address matches `ADDR_FROM` the sketch 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.

3.10.2.4 Configuration reference

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

3.10.2.4.1 Wi-Fi

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

Note

The Arduino UNO R4 WiFi only supports **2.4 GHz** Wi-Fi.

3.10.2.4.2 RPC endpoint — choose one provider

PublicNode (free, no signup, default):

```
#define RPC_HOST  "ethereum-sepolia-rpc.publicnode.com"
#define RPC_PORT  443
#define RPC_PATH  "/"
```

Infura (free tier, requires API key):

```
#define RPC_HOST        "sepolia.infura.io"
#define RPC_PORT        443
#define RPC_PROJECT_ID  "<YOUR_INFURA_PROJECT_ID>"
#define RPC_PATH        "/v3/" RPC_PROJECT_ID
#define RPC_API_SECRET  "<YOUR_INFURA_API_SECRET>"   // HTTP Basic Auth
```

3.10.2.4.3 TLS

Field	Default	Notes
WIFI_CA_CERT	ISRG Root X1 PEM (placeholder bundled in the sketch)	**Almost always needs overriding** — set it to the root CA your RPC provider uses
WIFI_DISABLE_CA_PINNING	unset	Defining it disables TLS validation — DEV ONLY

3.10.2.4.4 Wallet

Field	Example
CARD_PIN	"000000000" (must match cryptnox init)
CARD_PIN_LEN	9U (ASCII digit count)
ADDR_FROM	"4aadf6f331aea39e699db17c75a4b12d993956d2" (lowercase hex, no 0x)

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 sketch halts.

3.10.2.4.5 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

Field	Default	Notes
MAX_FEE	4000000000ULL	4 Gwei
GAS_LIMIT_ERC20	60000ULL	Standard ERC-20 transfer

3.10.2.5 Memory footprint

After setup.bat's memory-optimisation pass:

Section	Size
Flash (.text + .data)	~99 KB / 256 KB (38 %)
RAM (.bss + .data)	~11.6 KB / 32 KB (35 %)

The bulk of the flash (~25 KB) is the WiFi/HTTP stack plus the bundled CA certificate (~1.4 KB PEM). The Cryptnox SDK itself, together with the RLP / Keccak-256 helpers, sits at around ~20 KB.

3.10.2.6 Troubleshooting

Symptom	Cause	Fix
fetchNonce: POST failed, err=-1	TLS handshake fails (wrong CA pinned)	Re-extract the chain with openssl s_client -showcerts -servername HOST -connect HOST:443 </dev/null and put the root in WIFI_CA_CERT. For a quick diagnosis, temporarily set WIFI_DISABLE_CA_↩ PINNING to confirm the cert is the blocker
Connecting to WiFi.... (never stops)	Wrong SSID / password, or a 5 GHz-only network	UNO R4 WiFi is 2.4 GHz only — verify SSID, password, band
[fatal] 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
Tx broadcast OK but tef↩ PAST_NONCE on Etherscan	Stale nonce from the RPC	Wait a few seconds and re-run; the sketch 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

3.10.2.7 License

cryptnox-sdk-arduino is dual-licensed:

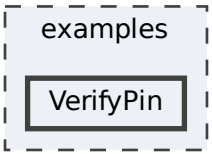
- **LGPL-3.0** for open-source projects and proprietary projects that comply with LGPL requirements

- **Commercial license** for projects that require a proprietary license without LGPL obligations (see COMMERCIAL.md for details)

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

3.11 examples/VerifyPin Directory Reference

Directory dependency graph for VerifyPin:



Files

- file VerifyPin.ino

3.11.1 Detailed Description

3.11.1.0.1 cryptnox-sdk-arduino

Arduino library for managing Cryptnox Hardware Wallet

3.11.2 VerifyPin — PIN Verification Over the Secure Channel

Open the Cryptnox Hardware Wallet secure channel and submit a PIN. The sketch 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 uploading.** 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 sketch halts on a wrong PIN — do not remove the `while (1)` guard, and verify `DEMO_PIN` matches the value used during `cryptnox init` **before** uploading.

3.11.2.1 Requirements

Component	Details
Hardware Wallet	Cryptnox Hardware Wallet, initialised with a known PIN
NFC reader	PN532 over SPI — see `hardware setup`
Board	Arduino UNO R4 Minima or WiFi
SDK	Installed via `setup.bat` — see `installation`

3.11.2.2 Quick start

1. **Edit `DEMO_PIN`** in `VerifyPin.ino` to match your card's PIN.
2. *File* → *Examples* → *CryptnoxWallet* → *VerifyPin*
3. *Tools* → *Board* → *Arduino UNO R4 Minima* (or *WiFi*) and *Tools* → *Port*
4. **Upload**, open the **Serial Monitor** at *115200 baud*, place the card on the antenna.

3.11.2.2.1 Expected output

Happy path (every second):

```
PIN accepted
```

Wrong PIN (sketch then halts):

```
Secured APDU: bad SW 0x63C2
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)

3.11.2.3 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`.

3.11.2.4 Step-by-step code

```
#define DEMO_PIN  "000000000"          // must match cryptnox init

CW_SecureSession session;
if (!wallet.connect(session)) {
    serialAdapter.println(F("Card not detected"));
    wallet.disconnect(session);
    delay(1000);
    return;
}

if (!wallet.verifyPin(session,
                      reinterpret_cast<const uint8_t*>(DEMO_PIN),
                      (uint8_t)strlen(DEMO_PIN))) {
    serialAdapter.println(F("PIN rejected -- halting to protect retry counter"));
    wallet.disconnect(session);
    while (1) {}                       // CRITICAL: do NOT loop on failure
}

serialAdapter.println(F("PIN accepted"));
wallet.disconnect(session);
```

3.11.2.5 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.

3.11.2.6 Troubleshooting

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

3.11.2.7 License

`cryptnox-sdk-arduino` 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

Namespace Documentation

4.1 patch_latex Namespace Reference

Functions

- _to_ascii (match)

Variables

- path = sys.argv[1]
- int title = sys.argv[2] if len(sys.argv) > 2 else None
- url
- write_to
- encoding
- s = _f.read()
- sty = os.path.join(os.path.dirname(path), "doxygen.sty")
- d = _f.read()
- idx = os.path.join(os.path.dirname(path), "index.tex")
- i = _f.read()
- keep
- tex
- body = _f.read()
- str marker = "\\begin{document}"
- split = body.find(marker)
- int head = -1 else ""
- int tail = -1 else body
- int fixed = head + re.sub(r"[^\x00-\x7F]", _to_ascii, tail)

4.1.1 Detailed Description

```
Post-process Doxygen's refman.tex before pdflatex, to match the Cryptnox PDF style.

Usage: python3 docs/patch_latex.py docs/latex/refman.tex ["Friendly Title"]
```

4.1.2 Function Documentation

4.1.2.1 _to_ascii()

```
patch_latex._to_ascii (
            match)  [protected]
```

Definition at line 176 of file patch_latex.py.

4.1.3 Variable Documentation

4.1.3.1 body

```
patch_latex.body = _f.read()
```
Definition at line 195 of file patch_latex.py.

4.1.3.2 d

```
patch_latex.d = _f.read()
```
Definition at line 138 of file patch_latex.py.

4.1.3.3 encoding

```
patch_latex.encoding
```
Definition at line 26 of file patch_latex.py.

4.1.3.4 fixed

```
int patch_latex.fixed = head + re.sub(r"[^\x00-\x7F]", _to_ascii, tail)
```
Definition at line 204 of file patch_latex.py.

4.1.3.5 head

```
int patch_latex.head = -1 else ""
```
Definition at line 202 of file patch_latex.py.

4.1.3.6 i

```
patch_latex.i = _f.read()
```
Definition at line 149 of file patch_latex.py.

4.1.3.7 idx

```
patch_latex.idx = os.path.join(os.path.dirname(path), "index.tex")
```
Definition at line 146 of file patch_latex.py.

4.1.3.8 keep

```
patch_latex.keep
```
Initial value:
```
00001 =  frozenset({
00002     0x00A0, 0x00B2, 0x00B3, 0x00D7,           # nbsp, superscript 2/3, multiplication
00003     0x2013, 0x2014, 0x2026,                   # en/em dash, ellipsis
00004     0x2018, 0x2019, 0x201C, 0x201D,           # curly quotes
00005     0x2190, 0x2192, 0x2194,                   # left/right/both arrows
00006     0x2264, 0x2265,                           # <= and >= (declared in the preamble)
00007 })
```
Definition at line 167 of file patch_latex.py.

4.1.3.9 marker

```
str patch_latex.marker = "\\begin{document}"
```
Definition at line 200 of file patch_latex.py.

4.1.3.10 path

```
patch_latex.path = sys.argv[1]
```
Definition at line 9 of file patch_latex.py.

4.1.3.11 s

```
patch_latex.s = _f.read()
```
Definition at line 27 of file patch_latex.py.

4.1.3.12 split

```
patch_latex.split = body.find(marker)
```
Definition at line 201 of file patch_latex.py.

4.1.3.13 sty

```
patch_latex.sty = os.path.join(os.path.dirname(path), "doxygen.sty")
```
Definition at line 135 of file patch_latex.py.

4.1.3.14 tail

```
int patch_latex.tail = -1 else body
```
Definition at line 203 of file patch_latex.py.

4.1.3.15 tex

```
patch_latex.tex
```
Definition at line 194 of file patch_latex.py.

4.1.3.16 title

```
int patch_latex.title = sys.argv[2] if len(sys.argv) > 2 else None
```
Definition at line 10 of file patch_latex.py.

4.1.3.17 url

```
patch_latex.url
```
Definition at line 22 of file patch_latex.py.

4.1.3.18 write_to

```
patch_latex.write_to
```
Definition at line 23 of file patch_latex.py.

Chapter 5

Class Documentation

5.1 __FlashStringHelper Class Reference

`#include <platform_compat.h>`

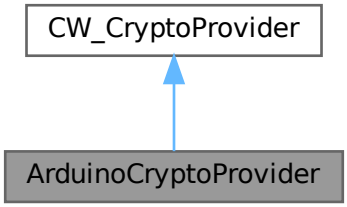
5.1.1 Detailed Description

Definition at line 41 of file platform_compat.h.
The documentation for this class was generated from the following file:

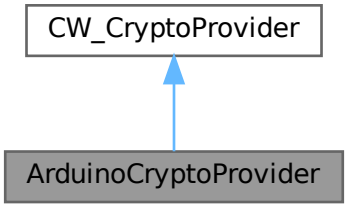
- src/cryptnox-sdk-cpp/platform_compat.h

5.2 ArduinoCryptoProvider Class Reference

`CW_CryptoProvider` implementation for the Arduino UNO R4 (RA4M1).
`#include <ArduinoCryptoProvider.h>`
Inheritance diagram for ArduinoCryptoProvider:



Collaboration diagram for ArduinoCryptoProvider:



Public Member Functions

- ArduinoCryptoProvider ()
  Construct the provider and install the RA4M1 TRNG into micro-ecc.
- ArduinoCryptoProvider (const ArduinoCryptoProvider &)=delete
- ArduinoCryptoProvider & operator= (const ArduinoCryptoProvider &)=delete

CW_CryptoProvider interface

- 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
  AES-CBC encrypt (selectable bit / null padding).
- uint16_t aesCbcDecrypt (uint8_t *in, uint16_t len, uint8_t *out, const uint8_t *key, uint8_t keyLen, uint8_t *iv, bool bitPadding) override
  AES-CBC decrypt (selectable bit / null padding).
- bool ecdh (const uint8_t *pubKey, const uint8_t *privKey, uint8_t *secret, CW_Curve curve) override
  Compute the ECDH shared secret on a portable curve identifier.
- bool makeKey (uint8_t *pubKey, uint8_t *privKey, CW_Curve curve) override
  Generate a fresh EC keypair via micro-ecc.
- bool random (uint8_t *dest, unsigned size) override
  Fill a buffer with random bytes from the RA4M1 hardware TRNG.
- bool ecdsaVerify (const uint8_t *pubKey64, const uint8_t *hash, size_t hashLen, const uint8_t *sig, CW_Curve curve) override
  Verify a raw r||s ECDSA signature against a message hash.

Public Member Functions inherited from CW_CryptoProvider

- virtual ~CW_CryptoProvider ()

Static Private Member Functions

- static const uECC_Curve_t * toUEccCurve (CW_Curve curve)
  Translate a portable CW_Curve to the matching micro-ecc descriptor.
- static uint8_t trngByte ()
  Generate one random byte from the RA4M1 hardware TRNG.
- static int trngCallback (uint8_t *dest, unsigned size)
  Static RNG callback registered with `uECC_set_rng()`.

Private Attributes

- AESLib _aes

  AESLib engine instance reused across all `aesCbc*` calls.

5.2.1 Detailed Description

`CW_CryptoProvider` implementation for the Arduino UNO R4 (RA4M1).
Provides the primitives the SDK needs to talk to a Cryptnox card:

Operation	Backing library / hardware
AES-CBC encrypt / decrypt	`AESLib`
SHA-256 (cert verification, optional via `CW_↩ VERIFY_CERT`)	`Crypto / SHA256`
SHA-512	`Crypto / SHA512`
ECDH + EC key generation + ECDSA verify	`micro-ecc`
Random bytes	RA4M1 on-chip True-RNG via the `trng` Arduino library

The constructor registers the internal static RNG callback with micro-ecc via `uECC_set_rng()` so the caller never needs to wire up randomness manually — without this, micro-ecc would fall back to its insecure default RNG and ECDH keypairs would be predictable.

Example

```
ArduinoCryptoProvider crypto;       // installs RA4M1 TRNG into micro-ecc
CW_CryptoProvider& provider = crypto;
uint8_t pub[64], priv[32];
provider.makeKey(pub, priv, CW_CURVE_SECP256R1);
```

Note

Non-copyable (RNG callback registration is global; cloning the provider has no useful semantics).

When `CW_VERIFY_CERT` is compiled to 0 the SHA-256 entry point is a no-op so the `SHA256` dependency can be dropped — the SDK only needs SHA-256 for card-certificate validation.

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 63 of file ArduinoCryptoProvider.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 ArduinoCryptoProvider() [1/2]

```
ArduinoCryptoProvider::ArduinoCryptoProvider ()
```

Construct the provider and install the RA4M1 TRNG into micro-ecc.
Construct the provider and register the TRNG-backed RNG with micro-ecc.
Definition at line 27 of file ArduinoCryptoProvider.cpp.
References trngCallback().
Referenced by ArduinoCryptoProvider(), and operator=().

5.2.2.2 ArduinoCryptoProvider() [2/2]

```
ArduinoCryptoProvider::ArduinoCryptoProvider (
            const ArduinoCryptoProvider & )  [delete]
```

References ArduinoCryptoProvider().

5.2.3 Member Function Documentation

5.2.3.1 aesCbcDecrypt()

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

AES-CBC decrypt (selectable bit / null padding).

Parameters

in,out	*in*	Ciphertext input buffer (may be mutated by AESLib).
in	*len*	Input length in bytes.
out	*out*	Plaintext output buffer.
in	*key*	Key bytes.
in	*keyLen*	Key length in bytes.
in,out	*iv*	IV in / next-block IV out.
in	*bitPadding*	`true` → strip ISO/IEC 7816-4 padding.

Returns

Number of plaintext bytes written into `out`.

Implements CW_CryptoProvider.
Definition at line 126 of file ArduinoCryptoProvider.cpp.
References _aes.

5.2.3.2 aesCbcEncrypt()

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

AES-CBC encrypt (selectable bit / null padding).

Parameters

in	*in*	Plaintext input buffer.
in	*len*	Input length in bytes.
out	*out*	Ciphertext output buffer (caller-allocated).
in	*key*	Key bytes.
in	*keyLen*	Key length in bytes (16, 24, or 32).
in,out	*iv*	IV in / next-block IV out (16 bytes).
in	*bitPadding*	`true` → ISO/IEC 7816-4 bit padding, `false` → null padding.

Returns

Number of ciphertext bytes written into `out`.

Implements CW_CryptoProvider.
Definition at line 115 of file ArduinoCryptoProvider.cpp.
References _aes.

5.2.3.3 ecdh()

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

Compute the ECDH shared secret on a portable curve identifier.
ECDH shared-secret computation via micro-ecc.
Translates `curve` to the matching micro-ecc curve descriptor (`uECC_secp256r1()` / `uECC_secp256k1()`) and calls `uECC_shared_secret()`.

Parameters

`in`	*pubKey*	Peer public key (X || Y, no leading 0x04).
`in`	*privKey*	Local private scalar.
`out`	*secret*	Output buffer for the shared X coordinate.
`in`	*curve*	Portable curve identifier (CW_CURVE_SECP256R1 or CW_CURVE_SECP256K1).

Returns

`true` on success, `false` if the peer's public key is not on the curve or `curve` is unrecognised.

Implements CW_CryptoProvider.
Definition at line 137 of file ArduinoCryptoProvider.cpp.
References toUEccCurve().

5.2.3.4 ecdsaVerify()

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

Verify a raw r||s ECDSA signature against a message hash.
ECDSA verify (raw r||s, 64 bytes) via micro-ecc.
Translates `curve` to a micro-ecc descriptor and calls `uECC_verify()`.

Parameters

`in`	*pubKey64*	Public key (X || Y, 64 bytes).
`in`	*hash*	Message hash buffer.
`in`	*hashLen*	Length of the hash in bytes (typically 32).
`in`	*sig*	Raw r||s signature (64 bytes).
`in`	*curve*	Portable curve identifier.

Returns

`true` if the signature verifies, `false` otherwise (bad signature, malformed key, or unrecognised curve).

Implements CW_CryptoProvider.
Definition at line 164 of file ArduinoCryptoProvider.cpp.
References toUEccCurve().

5.2.3.5 makeKey()

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

Generate a fresh EC keypair via micro-ecc.
Generate a fresh EC keypair via micro-ecc (uses the RA4M1 TRNG).
Uses the RA4M1 TRNG installed by the constructor — without that registration, `uECC_make_key()` would silently call its built-in fallback RNG and the resulting key would not be cryptographically secure.

Parameters

`out`	*pubKey*	Public key output (X || Y).
`out`	*privKey*	Private scalar output.
`in`	*curve*	Portable curve identifier.

Returns

`true` on success, `false` if `curve` is unrecognised or micro-ecc rejected the generated scalar (negligible probability — retry is safe).

Implements CW_CryptoProvider.
Definition at line 147 of file ArduinoCryptoProvider.cpp.
References toUEccCurve().

5.2.3.6 operator=()

```
ArduinoCryptoProvider & ArduinoCryptoProvider::operator= (
            const ArduinoCryptoProvider & )  [delete]
```

References ArduinoCryptoProvider().

5.2.3.7 random()

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

Fill a buffer with random bytes from the RA4M1 hardware TRNG.
Fill a buffer with hardware-TRNG random bytes.
Single source of randomness for the whole SDK.

Parameters

`out`	*dest*	Output buffer.
`in`	*size*	Number of bytes to write.

Returns

`true` if all `size` bytes were generated, `false` on a NULL pointer or zero size.

Implements CW_CryptoProvider.
Definition at line 157 of file ArduinoCryptoProvider.cpp.
References trngCallback().

5.2.3.8 sha256()

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

Compute SHA-256 over a contiguous buffer.
Compute SHA-256 (no-op when `CW_VERIFY_CERT` is 0 to drop the dependency).
Only does the work when `CW_VERIFY_CERT` is non-zero; otherwise the call short-circuits without touching `out` so the `SHA256` symbol is not pulled in by the linker.

Parameters

`in`	*data*	Pointer to the data to hash.
`in`	*len*	Length of `data` in bytes.
`out`	*out*	32-byte output buffer.

Returns

`true` on success (including the compiled-out fast path), `false` on a NULL argument.

Implements CW_CryptoProvider.
Definition at line 85 of file ArduinoCryptoProvider.cpp.

5.2.3.9 sha512()

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

Compute SHA-512 over a contiguous buffer.
Used by the secure channel layer for ECDH-derived session-key derivation; always compiled in.

Parameters

`in`	*data*	Pointer to the data to hash.
`in`	*len*	Length of `data` in bytes.
`out`	*out*	64-byte output buffer.

Returns

`true` on success, `false` on a NULL argument.

Implements CW_CryptoProvider.
Definition at line 103 of file ArduinoCryptoProvider.cpp.

5.2.3.10 toUEccCurve()

```
const uECC_Curve_t * ArduinoCryptoProvider::toUEccCurve (
```

```
            CW_Curve curve)  [static], [private]
```

Translate a portable CW_Curve to the matching micro-ecc descriptor.
Map a portable CW_Curve identifier to the matching micro-ecc descriptor.

Parameters

in	*curve*	Portable curve identifier.

Returns

micro-ecc curve descriptor, or NULL if curve is unknown.

uECC_secp256k1() is only available when micro-ecc is compiled with uECC_SUPPORTS_secp256k1=1. The Arduino memory-optimisation step (scripts\\enable_memory_optimization.bat) defines that to 0 to drop the curve and shave flash, so we guard the call — secp256k1 keys are still usable on the card itself (the card signs internally); the Arduino side only needs secp256r1 for the secure-channel ECDH.
Definition at line 41 of file ArduinoCryptoProvider.cpp.
References CW_CURVE_SECP256K1, and CW_CURVE_SECP256R1.
Referenced by ecdh(), ecdsaVerify(), and makeKey().

5.2.3.11 trngByte()

```
uint8_t ArduinoCryptoProvider::trngByte ()  [static], [private]
```

Generate one random byte from the RA4M1 hardware TRNG.
Read one random byte from the RA4M1 hardware TRNG (lazy init on first call).
Lazily initialises the TRNG global on first call.

Returns

A uniformly random byte in [0, 255].

Definition at line 54 of file ArduinoCryptoProvider.cpp.
Referenced by trngCallback().

5.2.3.12 trngCallback()

```
int ArduinoCryptoProvider::trngCallback (
            uint8_t * dest,
            unsigned size)  [static], [private]
```

Static RNG callback registered with uECC_set_rng().
Static RNG callback matching the micro-ecc uECC_RNG_Function signature.

Parameters

out	*dest*	Output buffer.
in	*size*	Number of bytes to produce.

Returns

1 on success, 0 on a NULL dest or zero size.

Definition at line 71 of file ArduinoCryptoProvider.cpp.
References trngByte().
Referenced by ArduinoCryptoProvider(), and random().

5.2.4 Member Data Documentation

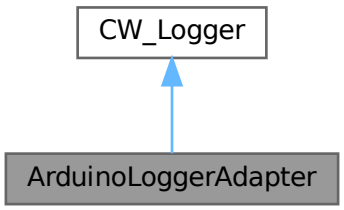
5.2.4.1 _aes

`AESLib ArduinoCryptoProvider::_aes [private]`
AESLib engine instance reused across all `aesCbc*` calls.
Definition at line 205 of file ArduinoCryptoProvider.h.
Referenced by aesCbcDecrypt(), and aesCbcEncrypt().
The documentation for this class was generated from the following files:

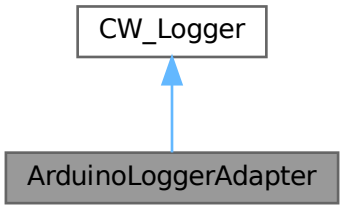
- src/ArduinoCryptoProvider.h
- src/ArduinoCryptoProvider.cpp

5.3 ArduinoLoggerAdapter Class Reference

`CW_Logger` implementation wrapping Arduino's `HardwareSerial`.
`#include <ArduinoLoggerAdapter.h>`
Inheritance diagram for ArduinoLoggerAdapter:



Collaboration diagram for ArduinoLoggerAdapter:



Public Member Functions

- ArduinoLoggerAdapter ()
  Construct an adapter that writes to the primary `Serial`.
- ArduinoLoggerAdapter (HardwareSerial *serial)
  Construct an adapter targeting a specific `HardwareSerial`.
- ~ArduinoLoggerAdapter () override=default

- ArduinoLoggerAdapter (const ArduinoLoggerAdapter &)=delete
- ArduinoLoggerAdapter & operator= (const ArduinoLoggerAdapter &)=delete

CW_Logger interface

- bool begin (unsigned long baudRate=115200UL) override
  Open the wrapped HardwareSerial at the given baud rate.
- void print (const __FlashStringHelper *str) override
  Forwards to `HardwareSerial::print(F-string)`.
- void print (const char *str) override
  Forwards to `HardwareSerial::print(const char*)`.
- void print (char c) override
  Forwards to `HardwareSerial::print(char)`.
- void print (uint8_t value, int base=DEC) override
  Forwards to `HardwareSerial::print(uint8_t, base)`.
- void print (uint16_t value, int base=DEC) override
  Forwards to `HardwareSerial::print(uint16_t, base)`.
- void print (uint32_t value, int base=DEC) override
  Forwards to `HardwareSerial::print(uint32_t, base)`.
- void print (int value, int base=DEC) override
  Forwards to `HardwareSerial::print(int, base)`.
- void println () override
  Emits a CR/LF.
- void println (const __FlashStringHelper *str) override
  Forwards then CR/LF.
- void println (const char *str) override
  Forwards then CR/LF.
- void println (char c) override
  Forwards then CR/LF.
- void println (uint8_t value, int base=DEC) override
  Forwards then CR/LF.
- void println (uint16_t value, int base=DEC) override
  Forwards then CR/LF.
- void println (uint32_t value, int base=DEC) override
  Forwards then CR/LF.
- void println (int value, int base=DEC) override
  Forwards then CR/LF.

Public Member Functions inherited from CW_Logger

- virtual ~CW_Logger ()

Private Attributes

- HardwareSerial * _serial
  Non-owning pointer to the wrapped HardwareSerial.

5.3.1 Detailed Description

`CW_Logger` implementation wrapping Arduino's `HardwareSerial`.
Lets CryptnoxWallet and CW_SecureChannel emit debug traces through the standard Arduino `Serial` API. By default the adapter wraps the primary `Serial` object, but any `HardwareSerial` instance can be injected — useful on the UNO R4 where `Serial1` is exposed on a separate connector.

Example

```
ArduinoLoggerAdapter logger;             // wraps Serial
ArduinoLoggerAdapter logger1(&Serial1);  // wraps Serial1
logger.begin(115200);
```

Warning

In production firmware, switch to NullLoggerAdapter — when `CW_DEBUG_LOGGING` is set the SDK emits raw APDUs and (briefly) PIN-related debug traces on the serial port (LOW-03). The NullLoggerAdapter route guarantees nothing reaches the UART regardless of compile-time flags.

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 48 of file ArduinoLoggerAdapter.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ArduinoLoggerAdapter() [1/3]

```
ArduinoLoggerAdapter::ArduinoLoggerAdapter ()
```

Construct an adapter that writes to the primary `Serial`.
Equivalent to `ArduinoLoggerAdapter(&Serial)`. The underlying serial port is not opened — call begin() once early in `setup()` with the desired baud rate before any log call.
Definition at line 17 of file ArduinoLoggerAdapter.cpp.
References _serial.
Referenced by ArduinoLoggerAdapter(), and operator=().

5.3.2.2 ArduinoLoggerAdapter() [2/3]

```
ArduinoLoggerAdapter::ArduinoLoggerAdapter (
            HardwareSerial * serial)  [explicit]
```

Construct an adapter targeting a specific `HardwareSerial`.

Parameters

in	*serial*	Pointer to the HardwareSerial to wrap (e.g. `&Serial1`). Must outlive the adapter; the adapter does not own the underlying object.

Definition at line 21 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.2.3 ~ArduinoLoggerAdapter()

```
ArduinoLoggerAdapter::~ArduinoLoggerAdapter ()  [override], [default]
```

5.3.2.4 ArduinoLoggerAdapter() [3/3]

```
ArduinoLoggerAdapter::ArduinoLoggerAdapter (
            const ArduinoLoggerAdapter & )  [delete]
```

References ArduinoLoggerAdapter().

5.3.3 Member Function Documentation

5.3.3.1 begin()

```
bool ArduinoLoggerAdapter::begin (
            unsigned long baudRate = 115200UL)  [override], [virtual]
```

Open the wrapped HardwareSerial at the given baud rate.
Thin wrapper over `HardwareSerial::begin()`. Must be called from the sketch's `setup()` before any log call.

Parameters

in	*baudRate*	Baud rate to open the port at (default 115200).

Returns

Always `true` — the underlying Arduino API has no failure channel.

Implements CW_Logger.
Definition at line 25 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.2 operator=()

```
ArduinoLoggerAdapter & ArduinoLoggerAdapter::operator= (
            const ArduinoLoggerAdapter & )  [delete]
```

References ArduinoLoggerAdapter(), and DEC.

5.3.3.3 print() [1/7]

```
void ArduinoLoggerAdapter::print (
            char c)  [override], [virtual]
```

Forwards to `HardwareSerial::print(char)`.
Implements CW_Logger.
Definition at line 32 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.4 print() [2/7]

```
void ArduinoLoggerAdapter::print (
            const __FlashStringHelper * str)  [override], [virtual]
```

Forwards to `HardwareSerial::print`(F-string).
Implements CW_Logger.
Definition at line 30 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.5 print() [3/7]

```
void ArduinoLoggerAdapter::print (
            const char * str)  [override], [virtual]
```

Forwards to `HardwareSerial::print(const char*)`.
Implements CW_Logger.
Definition at line 31 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.6 print() [4/7]

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

Forwards to `HardwareSerial::print(int, base)`.
Implements CW_Logger.
Definition at line 36 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.7 print() [5/7]

```
void ArduinoLoggerAdapter::print (
            uint16_t value,
```

```
            int base = DEC)  [override], [virtual]
```
Forwards to `HardwareSerial::print(uint16_t, base)`.
Implements CW_Logger.
Definition at line 34 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.8 print() [6/7]

```
void ArduinoLoggerAdapter::print (
            uint32_t value,
            int base = DEC)  [override], [virtual]
```
Forwards to `HardwareSerial::print(uint32_t, base)`.
Implements CW_Logger.
Definition at line 35 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.9 print() [7/7]

```
void ArduinoLoggerAdapter::print (
            uint8_t value,
            int base = DEC)  [override], [virtual]
```
Forwards to `HardwareSerial::print(uint8_t, base)`.
Implements CW_Logger.
Definition at line 33 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.10 println() [1/8]

```
void ArduinoLoggerAdapter::println ()  [override], [virtual]
```
Emits a CR/LF.
Implements CW_Logger.
Definition at line 38 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.11 println() [2/8]

```
void ArduinoLoggerAdapter::println (
            char c)  [override], [virtual]
```
Forwards then CR/LF.
Implements CW_Logger.
Definition at line 41 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.12 println() [3/8]

```
void ArduinoLoggerAdapter::println (
            const __FlashStringHelper * str)  [override], [virtual]
```
Forwards then CR/LF.
Implements CW_Logger.
Definition at line 39 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.13 println() [4/8]

```
void ArduinoLoggerAdapter::println (
            const char * str)  [override], [virtual]
```
Forwards then CR/LF.
Implements CW_Logger.
Definition at line 40 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.14 println() [5/8]

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

Forwards then CR/LF.
Implements CW_Logger.
Definition at line 45 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.15 println() [6/8]

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

Forwards then CR/LF.
Implements CW_Logger.
Definition at line 43 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.16 println() [7/8]

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

Forwards then CR/LF.
Implements CW_Logger.
Definition at line 44 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.3.17 println() [8/8]

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

Forwards then CR/LF.
Implements CW_Logger.
Definition at line 42 of file ArduinoLoggerAdapter.cpp.
References _serial.

5.3.4 Member Data Documentation

5.3.4.1 _serial

```
HardwareSerial* ArduinoLoggerAdapter::_serial  [private]
```

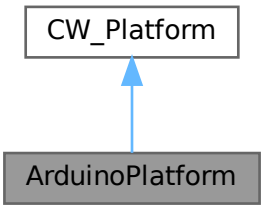
Non-owning pointer to the wrapped HardwareSerial.
Definition at line 107 of file ArduinoLoggerAdapter.h.
Referenced by ArduinoLoggerAdapter(), ArduinoLoggerAdapter(), begin(), print(), print(), print(), print(), print(), print(), print(), println(), println(), println(), println(), println(), println(), println(), and println().
The documentation for this class was generated from the following files:

- src/ArduinoLoggerAdapter.h
- src/ArduinoLoggerAdapter.cpp

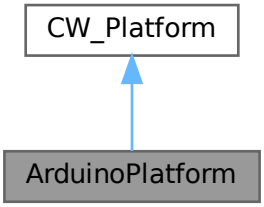
5.4 ArduinoPlatform Class Reference

`CW_Platform` implementation using Arduino's blocking `delay()`.
`#include <ArduinoPlatform.h>`

Inheritance diagram for ArduinoPlatform:



Collaboration diagram for ArduinoPlatform:



Public Member Functions

- ArduinoPlatform ()=default
- ~ArduinoPlatform () override=default
- ArduinoPlatform (const ArduinoPlatform &)=delete
- ArduinoPlatform & operator= (const ArduinoPlatform &)=delete
- void sleep_ms (uint32_t ms) override

  Block for `ms` milliseconds via Arduino's `delay()`.

Public Member Functions inherited from CW_Platform

- virtual ~CW_Platform ()

5.4.1 Detailed Description

`CW_Platform` implementation using Arduino's blocking `delay()`.
Single-method adapter — exists only so the SDK can stay platform-independent. Construct one alongside the other adapters and pass it as the 4th argument of CryptnoxWallet's constructor.

Example

```
ArduinoPlatform       platform;
ArduinoCryptoProvider crypto;
NullLoggerAdapter     logger;
PN532Adapter          nfc(logger, 10);
CryptnoxWallet        wallet(nfc, logger, crypto, platform);
```

Note

Blocking — sleep_ms parks the calling task until the timeout expires. On bare-metal Arduino this is the expected behaviour; on cooperative-scheduler ports a different platform implementation (e.g. one that yields) is appropriate.

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 49 of file ArduinoPlatform.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 ArduinoPlatform() [1/2]

```
ArduinoPlatform::ArduinoPlatform ()  [default]
```

Referenced by ArduinoPlatform(), and operator=().

5.4.2.2 ~ArduinoPlatform()

```
ArduinoPlatform::~ArduinoPlatform ()  [override], [default]
```

5.4.2.3 ArduinoPlatform() [2/2]

```
ArduinoPlatform::ArduinoPlatform (
            const ArduinoPlatform & )  [delete]
```

References ArduinoPlatform().

5.4.3 Member Function Documentation

5.4.3.1 operator=()

```
ArduinoPlatform & ArduinoPlatform::operator= (
            const ArduinoPlatform & )  [delete]
```

References ArduinoPlatform().

5.4.3.2 sleep_ms()

```
void ArduinoPlatform::sleep_ms (
            uint32_t ms)  [override], [virtual]
```

Block for `ms` milliseconds via Arduino's `delay()`.
Forward to Arduino's `delay()`.

Parameters

`in`	*ms*	Duration to sleep, in milliseconds.

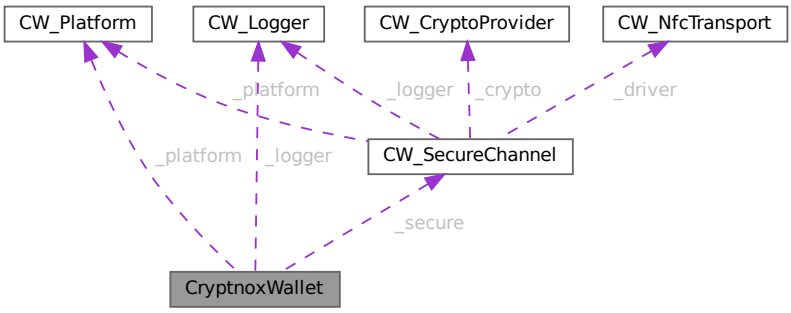
Implements CW_Platform.
Definition at line 16 of file ArduinoPlatform.cpp.
The documentation for this class was generated from the following files:

- src/ArduinoPlatform.h
- src/ArduinoPlatform.cpp

5.5 CryptnoxWallet Class Reference

High-level interface for interacting with a Cryptnox Hardware Wallet over NFC.
`#include <CryptnoxWallet.h>`

Collaboration diagram for CryptnoxWallet:



Public Member Functions

- CryptnoxWallet (CW_NfcTransport &driver, CW_Logger &logger, CW_CryptoProvider &crypto, CW_Platform &platform)

  Construct a CryptnoxWallet.
- CryptnoxWallet (const CryptnoxWallet &)=delete
- CryptnoxWallet & operator= (const CryptnoxWallet &)=delete
- bool begin ()

  Initialize the NFC module via the underlying transport driver.
- bool connect (CW_SecureSession &session)

  Connect to the Cryptnox card and establish a secure channel.
- bool establishSecureChannel (CW_SecureSession &session)

  Establish a secure channel (SELECT → certificate → ECDH → mutual auth).
- void disconnect (CW_SecureSession &session)

  Disconnect and securely clear the session.
- bool getCardInfo (CW_SecureSession &session, CW_CardInfo *info=NULL)

  Send a secured GET CARD INFO APDU (0x80FA0000) and optionally decode the owner name/email from the response.
- bool verifyPin (CW_SecureSession &session, const uint8_t *pin, uint8_t pinLength)

  Verify the PIN code on the card.
- CW_SignResult sign (CW_SignRequest &request)

  Sign a 32-byte digest using a card-resident key.
- bool writeUserData (CW_SecureSession &session, uint8_t slot, const uint8_t *data, uint16_t dataLength)

  Write data to a user memory slot, paginating in CW_USER_DATA_PAGE_SIZE chunks.

Static Public Member Functions

- static bool parseDerSignature (const uint8_t *der, uint8_t derLength, uint8_t *r, uint8_t &rLength, uint8_t *s, uint8_t &sLength)

  Parse a DER-encoded ECDSA signature to extract raw r and s values.

Private Member Functions

- bool isSecureChannelOpen (const CW_SecureSession &session) const
- bool printPN532FirmwareVersion ()
- bool validateSignRequest (const CW_SignRequest &request, CW_SignResult &result)

- void buildSignPayload (const CW_SignRequest &request, uint8_t *data, uint16_t &dataLength)
- bool sendSignApdu (CW_SignRequest &request, const uint8_t *data, uint16_t dataLength, uint8_t *derResponse, uint16_t &derLength, CW_SignResult &result)
- bool extractRawSignature (const uint8_t *derResponse, uint16_t derLength, CW_SignResult &result)
- void debugPrintSignature (const uint8_t *signature)

Private Attributes

- CW_Logger & _logger

  Logging interface.
- CW_Platform & _platform

  Platform abstraction (sleep_ms).
- CW_SecureChannel _secure

  Owned secure channel.

5.5.1 Detailed Description

High-level interface for interacting with a Cryptnox Hardware Wallet over NFC.
Manages card connection, secure channel establishment (delegated to CW_SecureChannel), PIN verification, transaction signing, user-data writing, and card-info retrieval.
Dependencies are injected by the caller via the constructor. The class itself only talks to the four abstract adapters, keeping the implementation platform-independent.

Typical lifecycle

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

CW_SecureSession session;
if (wallet.connect(session)) {
    wallet.verifyPin(session, pin, pinLen);
    wallet.sign(req);
}
wallet.disconnect(session);   // mandatory -- even on connect() failure
```

Note

Single-shot use — the class is non-copyable. Reuse the same CryptnoxWallet instance across multiple card sessions; do not construct one per APDU.

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 160 of file CryptnoxWallet.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 CryptnoxWallet() [1/2]

```
CryptnoxWallet::CryptnoxWallet (
            CW_NfcTransport & driver,
            CW_Logger & logger,
            CW_CryptoProvider & crypto,
            CW_Platform & platform)
```

Construct a CryptnoxWallet.

Parameters

driver	Reference to the NFC transport implementation.

logger	Reference to the logging implementation.
crypto	Reference to the crypto provider implementation.
platform	Reference to the platform abstraction (for sleep_ms).

Definition at line 27 of file CryptnoxWallet.cpp.
References _logger, _platform, _secure, and platform.
Referenced by CryptnoxWallet(), and operator=().

5.5.2.2 CryptnoxWallet() [2/2]

```
CryptnoxWallet::CryptnoxWallet (
            const CryptnoxWallet & )  [delete]
```

References CryptnoxWallet().

5.5.3 Member Function Documentation

5.5.3.1 begin()

```
bool CryptnoxWallet::begin ()
```

Initialize the NFC module via the underlying transport driver.

Returns

true if the module was successfully initialised, false otherwise.

Definition at line 36 of file CryptnoxWallet.cpp.
References _secure, and printPN532FirmwareVersion().

5.5.3.2 buildSignPayload()

```
void CryptnoxWallet::buildSignPayload (
            const CW_SignRequest & request,
            uint8_t * data,
            uint16_t & dataLength)  [private]
```

Definition at line 431 of file CryptnoxWallet.cpp.
References CW_HASH_SIZE, CW_MAX_DERIVE_PATH_LENGTH, CW_MAX_PIN_LENGTH, CW_SIGN_DERIVE_K1, CW_SIGN_DERIVE_R1, CW_SignRequest::derivePath, CW_SignRequest::derivePathLength, CW_SignRequest::hash, CW_SignRequest::hashLength, CW_SignRequest::keyType, CW_SignRequest::pin, CW_SignRequest::pinLessMode, and CW_Utils::safe_memcpy().
Referenced by sign().

5.5.3.3 connect()

```
bool CryptnoxWallet::connect (
            CW_SecureSession & session)
```

Connect to the Cryptnox card and establish a secure channel.
Retries the full card activation sequence up to `CW_CONNECT_MAX_ATTEMPTS` times. On any failure (including transient transport errors) the session is securely wiped before the next attempt so no partial key material can survive a retry (CRIT-04).

Parameters

`out`	*session*	Secure session to populate with derived keys and IV on success; left zero-wiped on failure.

Returns

true if the secure channel was established and `session` is ready for use, false otherwise.

Postcondition

On true: `session` holds valid Kenc / Kmac / IV.

On false: `session` is zero-wiped.

Warning

Always call disconnect() after this — even on failure — to release the reader for the next card cycle.

Definition at line 44 of file CryptnoxWallet.cpp.
References _logger, _platform, _secure, CW_SecureSession::clear(), CW_CONNECT_MAX_ATTEMPTS, establishSecureChannel(), and F.

5.5.3.4 debugPrintSignature()

```
void CryptnoxWallet::debugPrintSignature (
            const uint8_t * signature)  [private]
```

Definition at line 543 of file CryptnoxWallet.cpp.
References _logger, CW_RAW_SIGNATURE_SIZE, F, and HEX.
Referenced by sign().

5.5.3.5 disconnect()

```
void CryptnoxWallet::disconnect (
            CW_SecureSession & session)
```

Disconnect and securely clear the session.
Wipes any session keys and resets the NFC reader so the next card detection cycle starts from a clean state.

Parameters

`in,out`	*session*	Session to clear. Safe to pass a never-connected or partially-connected session.

Precondition

Must be called at the end of every card-processing iteration — including iterations where connect() failed — otherwise the NFC reader may remain in an unresponsive state.

Definition at line 158 of file CryptnoxWallet.cpp.
References _secure, CW_SecureSession::clear(), and isSecureChannelOpen().

5.5.3.6 establishSecureChannel()

```
bool CryptnoxWallet::establishSecureChannel (
            CW_SecureSession & session)
```

Establish a secure channel (SELECT → certificate → ECDH → mutual auth).
Lower-level than connect(): runs the full activation sequence once, without the retry loop. Used internally by connect(); exposed for advanced callers that handle retry policy themselves.

Parameters

out	*session*	Secure session to populate.

Returns

true if mutual authentication succeeded, false if any step of the activation sequence (SELECT, certificate chain verification, ECDH, MAC check) failed.

Warning

All sensitive ephemeral key material is wiped from the stack on every exit path (H-01, M-02).

Definition at line 75 of file CryptnoxWallet.cpp.
References _logger, _secure, CW_CERT_OK, CW_CURVE_SECP256R1, F, HEX, and CW_Utils::secure_wipe().
Referenced by connect().

5.5.3.7 extractRawSignature()

```
bool CryptnoxWallet::extractRawSignature (
            const uint8_t * derResponse,
            uint16_t derLength,
            CW_SignResult & result)  [private]
```

Definition at line 480 of file CryptnoxWallet.cpp.
References _logger, CW_DER_TAG_SEQUENCE, CW_NOK, CW_RAW_SIGNATURE_SIZE, CW_SignResult::errorCode, F, parseDerSignature(), CW_Utils::safe_memcpy(), CW_Utils::secure_wipe(), and CW_SignResult::signature.
Referenced by sign().

5.5.3.8 getCardInfo()

```
bool CryptnoxWallet::getCardInfo (
            CW_SecureSession & session,
            CW_CardInfo * info = NULL)
```

Send a secured GET CARD INFO APDU (0x80FA0000) and optionally decode the owner name/email from the response.

Parameters

in,out	*session*	Valid secure session.
out	*info*	Optional output. When non-NULL and the call succeeds, populated with the card's owner name and email (ASCII, NUL-terminated).

Returns

true if the secure exchange completed and (when info is non-NULL) parsing the name/email fields succeeded.

Definition at line 165 of file CryptnoxWallet.cpp.
References _logger, _secure, CW_CARD_EMAIL_MAX_LEN, CW_CARD_NAME_MAX_LEN, CW_CardInfo::email, F, isSecureChannelOpen(), CW_CardInfo::name, CW_Utils::safe_memcpy(), and CW_Utils::secure_wipe().

5.5.3.9 isSecureChannelOpen()

```
bool CryptnoxWallet::isSecureChannelOpen (
            const CW_SecureSession & session) const  [private]
```

Definition at line 370 of file CryptnoxWallet.cpp.
References CW_SecureSession::aesKey, and CW_AESKEY_SIZE.
Referenced by disconnect(), getCardInfo(), validateSignRequest(), verifyPin(), and writeUserData().

5.5.3.10 operator=()

```
CryptnoxWallet & CryptnoxWallet::operator= (
            const CryptnoxWallet & )  [delete]
```

References CryptnoxWallet().

5.5.3.11 parseDerSignature()

```
bool CryptnoxWallet::parseDerSignature (
            const uint8_t * der,
            uint8_t derLength,
            uint8_t * r,
            uint8_t & rLength,
            uint8_t * s,
            uint8_t & sLength)  [static]
```

Parse a DER-encoded ECDSA signature to extract raw r and s values.

Parameters

in	*der*	DER-encoded signature bytes.
in	*derLength*	DER length.
out	*r*	Buffer for r (at least 33 bytes).
out	*rLength*	Actual r length written.
out	*s*	Buffer for s (at least 33 bytes).
out	*sLength*	Actual s length written.

Returns

true on success, false on malformed DER.

Definition at line 322 of file CryptnoxWallet.cpp.
References CW_DER_TAG_INTEGER, CW_DER_TAG_SEQUENCE, and CW_Utils::safe_memcpy().
Referenced by extractRawSignature().

5.5.3.12 printPN532FirmwareVersion()

```
bool CryptnoxWallet::printPN532FirmwareVersion ()  [private]
```

Definition at line 378 of file CryptnoxWallet.cpp.
References _secure.
Referenced by begin().

5.5.3.13 sendSignApdu()

```
bool CryptnoxWallet::sendSignApdu (
            CW_SignRequest & request,
            const uint8_t * data,
            uint16_t dataLength,
            uint8_t * derResponse,
            uint16_t & derLength,
            CW_SignResult & result)  [private]
```

Definition at line 456 of file CryptnoxWallet.cpp.
References _logger, _secure, CW_SIGN_NO_KEY_LOADED, CW_SignResult::errorCode, F, CW_SignRequest::keyType, CW_SignRequest::session, and CW_SignRequest::signatureType.
Referenced by sign().

5.5.3.14 sign()

```
CW_SignResult CryptnoxWallet::sign (
            CW_SignRequest & request)
```

Sign a 32-byte digest using a card-resident key.
Builds the SIGN payload (hash || optional BIP32 path || optional PIN), sends it through the secure channel, parses the DER signature returned by the card, and unpacks it into the canonical 64-byte raw form (r[32] || s[32]).

Parameters

`in`	*request*	Sign parameters — must reference a valid secure session, a 32-byte hash, the desired key/signature type, and the PIN (unless `pinLessMode` is set).

Returns

CW_SignResult. On success `errorCode` is CW_OK and `signature` holds the 64-byte raw signature. On failure the signature is zeroed and `errorCode` indicates the cause:

Return values

CW_OK	Signature valid.
CW_INVALID_SESSION	Secure channel not open.
CW_SIGN_KEY_TOO_SHORT	Bad hash buffer / length.
CW_SIGN_NO_KEY_LOADED	Card rejected the SIGN APDU.
CW_SIGN_PIN_INCORRECT	PIN length out of range.
CW_SIGN_KEY_TOO_SHORT_WITH_PINLESS_MODE	PIN-less mode requested but `keyType` is not CW_SIGN_PINLESS_K1.

Warning

When `pinLessMode` is false the `request.pin` field must be populated. The destructor of CW_SignRequest securely wipes the PIN, but the caller must zero any other copy.

Definition at line 293 of file CryptnoxWallet.cpp.
References buildSignPayload(), CW_HASH_SIZE, CW_MAX_DERIVE_PATH_LENGTH, CW_MAX_PIN_LENGTH, CW_OK, debugPrintSignature(), CW_SignResult::errorCode, extractRawSignature(), CW_Utils::secure_wipe(), sendSignApdu(), CW_SignResult::signature, and validateSignRequest().

5.5.3.15 validateSignRequest()

```
bool CryptnoxWallet::validateSignRequest (
            const CW_SignRequest & request,
            CW_SignResult & result)  [private]
```

Definition at line 382 of file CryptnoxWallet.cpp.
References _logger, CW_HASH_SIZE, CW_INVALID_SESSION, CW_MAX_PIN_LENGTH, CW_MIN_PIN_LENGTH, CW_SIGN_KEY_TOO_SHORT, CW_SIGN_KEY_TOO_SHORT_WITH_PINLESS_MODE, CW_SIGN_PIN_INCORRECT, CW_SIGN_PINLESS_K1, CW_SignResult::errorCode, F, CW_SignRequest::hash, CW_SignRequest::hashLength, isSecureChannelOpen(), CW_SignRequest::keyType, CW_SignRequest::pin, CW_SignRequest::pinLessMode, and CW_SignRequest::session.
Referenced by sign().

5.5.3.16 verifyPin()

```
bool CryptnoxWallet::verifyPin (
            CW_SecureSession & session,
            const uint8_t * pin,
            uint8_t pinLength)
```

Verify the PIN code on the card.
Sends an encrypted VERIFY PIN APDU. The card maintains a try counter: every wrong attempt decrements it, and reaching zero locks the PIN permanently until a successful PUK / re-initialisation flow.

Parameters

in,out	*session*	Valid secure session.
in	*pin*	PIN bytes (ASCII digits, 4–9 characters).
in	*pinLength*	Length of the PIN (must be in [CW_MIN_PIN_LENGTH, CW_MAX_PIN_LENGTH]).

Returns

true if the card accepted the PIN, false on wrong PIN, closed or invalid session, length out of range, or transport / MAC failure.

Warning

Each failed attempt decrements the card's PIN counter. Treat a false return as "wrong PIN" only after confirming session validity — a transport glitch should not be retried with a new PIN.

Definition at line 219 of file CryptnoxWallet.cpp.
References _logger, _secure, CW_MAX_PIN_LENGTH, CW_MIN_PIN_LENGTH, F, isSecureChannelOpen(), CW_Utils::safe_memcpy(), and CW_Utils::secure_wipe().

5.5.3.17 writeUserData()

```
bool CryptnoxWallet::writeUserData (
            CW_SecureSession & session,
            uint8_t slot,
            const uint8_t * data,
            uint16_t dataLength)
```

Write data to a user memory slot, paginating in CW_USER_DATA_PAGE_SIZE chunks.

Parameters

in,out	*session*	Valid secure session.
in	*slot*	User data slot index.
in	*data*	Data to write.
in	*dataLength*	Total bytes to write.

Returns

true if all pages written successfully, false otherwise.

Definition at line 241 of file CryptnoxWallet.cpp.
References _logger, _secure, CW_USER_DATA_PAGE_SIZE, F, and isSecureChannelOpen().

5.5.4 Member Data Documentation

5.5.4.1 _logger

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

Logging interface.
Definition at line 328 of file CryptnoxWallet.h.

Referenced by connect(), CryptnoxWallet(), debugPrintSignature(), establishSecureChannel(), extractRawSignature(), getCardInfo(), sendSignApdu(), validateSignRequest(), verifyPin(), and writeUserData().

5.5.4.2 _platform

```
CW_Platform& CryptnoxWallet::_platform  [private]
```
Platform abstraction (sleep_ms).
Definition at line 329 of file CryptnoxWallet.h.
Referenced by connect(), and CryptnoxWallet().

5.5.4.3 _secure

```
CW_SecureChannel CryptnoxWallet::_secure  [private]
```
Owned secure channel.
Definition at line 330 of file CryptnoxWallet.h.
Referenced by begin(), connect(), CryptnoxWallet(), disconnect(), establishSecureChannel(), getCardInfo(), printPN532FirmwareVersion(), sendSignApdu(), verifyPin(), and writeUserData().
The documentation for this class was generated from the following files:

- src/cryptnox-sdk-cpp/CryptnoxWallet.h
- src/cryptnox-sdk-cpp/CryptnoxWallet.cpp

5.6 CW_CardInfo Struct Reference

Subset of the Cryptnox card info returned by APDU 0x80FA0000.
`#include <CryptnoxWallet.h>`

Public Member Functions

- CW_CardInfo ()

Public Attributes

- char name [CW_CARD_NAME_MAX_LEN+1U]
- char email [CW_CARD_EMAIL_MAX_LEN+1U]

5.6.1 Detailed Description

Subset of the Cryptnox card info returned by APDU 0x80FA0000.
Mirrors the fields the Python SDK exposes as `card._owner`: ASCII name and email programmed when the card was initialised. NUL-terminated.

Examples

Connect.ino.

Definition at line 55 of file CryptnoxWallet.h.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 CW_CardInfo()

```
CW_CardInfo::CW_CardInfo ()  [inline]
```
Definition at line 59 of file CryptnoxWallet.h.
References email, and name.

5.6.3 Member Data Documentation

5.6.3.1 email

`char CW_CardInfo::email[CW_CARD_EMAIL_MAX_LEN+1U]`
NUL-terminated ASCII email.

Examples

Connect.ino.

Definition at line 57 of file CryptnoxWallet.h.
Referenced by CW_CardInfo(), CryptnoxWallet::getCardInfo(), and loop().

5.6.3.2 name

`char CW_CardInfo::name[CW_CARD_NAME_MAX_LEN+1U]`
NUL-terminated ASCII name.

Examples

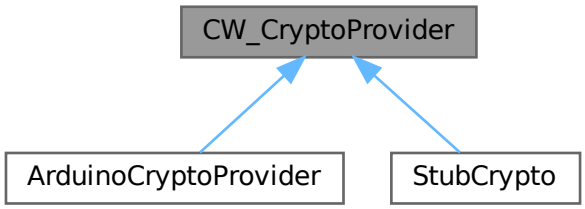
Connect.ino.

Definition at line 56 of file CryptnoxWallet.h.
Referenced by CW_CardInfo(), CryptnoxWallet::getCardInfo(), and loop().
The documentation for this struct was generated from the following file:

- src/cryptnox-sdk-cpp/CryptnoxWallet.h

5.7 CW_CryptoProvider Class Reference

Abstract interface for cryptographic operations used by CW_SecureChannel.
`#include <CW_CryptoProvider.h>`
Inheritance diagram for CW_CryptoProvider:



Public Member Functions

- virtual bool sha256 (const uint8_t *data, size_t len, uint8_t *out)=0
  Compute SHA-256 over a contiguous data buffer.
- virtual bool sha512 (const uint8_t *data, size_t len, uint8_t *out)=0
  Compute SHA-512 over a contiguous data buffer.
- virtual 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)=0
  AES-CBC encrypt.
- virtual uint16_t aesCbcDecrypt (uint8_t *in, uint16_t len, uint8_t *out, const uint8_t *key, uint8_t keyLen, uint8_t *iv, bool bitPadding)=0

AES-CBC decrypt.

- virtual bool ecdh (const uint8_t *pubKey, const uint8_t *privKey, uint8_t *secret, CW_Curve curve)=0

  ECDH shared secret computation.
- virtual bool makeKey (uint8_t *pubKey, uint8_t *privKey, CW_Curve curve)=0

  Generate a new EC key pair.
- virtual bool random (uint8_t *dest, unsigned size)=0

  Fill a buffer with cryptographically random bytes.
- virtual bool ecdsaVerify (const uint8_t *pubKey64, const uint8_t *hash, size_t hashLen, const uint8_t *sig, CW_Curve curve)=0

  Verify an ECDSA signature (raw r||s, 64 bytes) against a message hash.
- virtual ~CW_CryptoProvider ()

5.7.1 Detailed Description

Abstract interface for cryptographic operations used by CW_SecureChannel.
Decouples the secure channel protocol from any specific crypto library. The concrete ESP32 implementation (ESP32CryptoProvider) wraps mbedTLS (SHA-256/SHA-512/AES-CBC hardware-accelerated on ESP32-S3) and the ESP32 hardware TRNG for random number generation.
Definition at line 45 of file CW_CryptoProvider.h.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 ~CW_CryptoProvider()

```
virtual CW_CryptoProvider::~CW_CryptoProvider ()  [inline], [virtual]
```

Definition at line 146 of file CW_CryptoProvider.h.

5.7.3 Member Function Documentation

5.7.3.1 aesCbcDecrypt()

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

AES-CBC decrypt.

Parameters

in	*in*	Ciphertext input buffer (non-const; may be modified internally).
in	*len*	Length of the ciphertext.
out	*out*	Plaintext output buffer.
in	*key*	AES key bytes.
in	*keyLen*	AES key length in bytes.
in,out	*iv*	16-byte IV used as decrypt IV.
in	*bitPadding*	true = Bit padding removal; false = Null padding (no removal).

Returns

Length of the plaintext written to out.

Implemented in ArduinoCryptoProvider, and StubCrypto.

5.7.3.2 aesCbcEncrypt()

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

AES-CBC encrypt.

Parameters

in	*in*	Plaintext input buffer.
in	*len*	Length of the plaintext.
out	*out*	Ciphertext output buffer (must be large enough for padding).
in	*key*	AES key bytes.
in	*keyLen*	AES key length in bytes (16, 24, or 32).
in,out	*iv*	16-byte IV; updated to last cipher block on return.
in	*bitPadding*	true = ISO/IEC 9797-1 Method 2 (Bit) padding; false = Null padding (no padding added).

Returns

Length of the ciphertext written to `out`.

Implemented in ArduinoCryptoProvider, and StubCrypto.

5.7.3.3 ecdh()

```
virtual bool CW_CryptoProvider::ecdh (
            const uint8_t * pubKey,
            const uint8_t * privKey,
            uint8_t * secret,
            CW_Curve curve)  [pure virtual]
```

ECDH shared secret computation.

Parameters

in	*pubKey*	Remote public key (64 bytes, X||Y, no 0x04 prefix).
in	*privKey*	Local private key (32 bytes).
out	*secret*	32-byte shared secret output.
in	*curve*	Curve identifier (CW_CURVE_SECP256R1 or CW_CURVE_SECP256K1).

Returns

true on success, false otherwise.

Implemented in ArduinoCryptoProvider, and StubCrypto.

5.7.3.4 ecdsaVerify()

```
virtual bool CW_CryptoProvider::ecdsaVerify (
            const uint8_t * pubKey64,
            const uint8_t * hash,
```

```
            size_t hashLen,
            const uint8_t * sig,
            CW_Curve curve)  [pure virtual]
```

Verify an ECDSA signature (raw r||s, 64 bytes) against a message hash.

Parameters

in	*pubKey64*	64-byte public key (X||Y, no 0x04 prefix).
in	*hash*	Message hash buffer.
in	*hashLen*	Length of the hash in bytes.
in	*sig*	64-byte raw signature (r[32]||s[32]).
in	*curve*	Curve identifier for the verification operation.

Returns

true if the signature is valid, false otherwise.

Implemented in ArduinoCryptoProvider, and StubCrypto.

5.7.3.5 makeKey()

```
virtual bool CW_CryptoProvider::makeKey (
            uint8_t * pubKey,
            uint8_t * privKey,
            CW_Curve curve)  [pure virtual]
```

Generate a new EC key pair.

Parameters

out	*pubKey*	64-byte public key output (X||Y, no prefix).
out	*privKey*	32-byte private key output.
in	*curve*	Curve identifier (CW_CURVE_SECP256R1 or CW_CURVE_SECP256K1).

Returns

true on success, false otherwise.

Implemented in ArduinoCryptoProvider, and StubCrypto.

5.7.3.6 random()

```
virtual bool CW_CryptoProvider::random (
            uint8_t * dest,
            unsigned size)  [pure virtual]
```

Fill a buffer with cryptographically random bytes.

Parameters

out	*dest*	Buffer to fill.
in	*size*	Number of bytes to generate.

Returns

true on success, false otherwise.

Implemented in ArduinoCryptoProvider, and StubCrypto.

5.7.3.7 sha256()

```
virtual bool CW_CryptoProvider::sha256 (
            const uint8_t * data,
            size_t len,
            uint8_t * out)  [pure virtual]
```

Compute SHA-256 over a contiguous data buffer.

Parameters

in	*data*	Input buffer.
in	*len*	Number of bytes to hash.
out	*out*	32-byte output buffer.

Returns

true on success, false if the underlying hash accelerator faults.

Implemented in ArduinoCryptoProvider, and StubCrypto.

5.7.3.8 sha512()

```
virtual bool CW_CryptoProvider::sha512 (
            const uint8_t * data,
            size_t len,
            uint8_t * out)  [pure virtual]
```

Compute SHA-512 over a contiguous data buffer.

Parameters

in	*data*	Input buffer.
in	*len*	Number of bytes to hash.
out	*out*	64-byte output buffer.

Returns

true on success, false if the underlying hash accelerator faults.

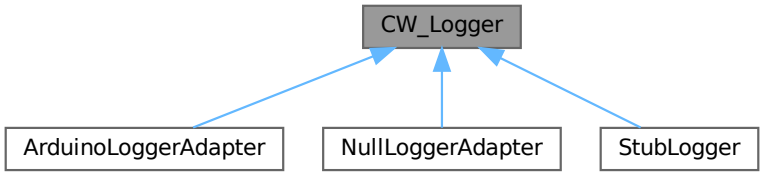
Implemented in ArduinoCryptoProvider, and StubCrypto.
The documentation for this class was generated from the following file:

- src/cryptnox-sdk-cpp/CW_CryptoProvider.h

5.8 CW_Logger Class Reference

Abstract interface for serial/debug output.
`#include <CW_Logger.h>`

Inheritance diagram for CW_Logger:



Public Member Functions

- virtual bool begin (unsigned long baudRate=115200UL)=0

  Initialize the logging interface.
- virtual ~CW_Logger ()

Print methods (no newline)

- virtual void print (const __FlashStringHelper *str)=0
- virtual void print (const char *str)=0
- virtual void print (char c)=0
- virtual void print (uint8_t value, int base=DEC)=0
- virtual void print (uint16_t value, int base=DEC)=0
- virtual void print (uint32_t value, int base=DEC)=0
- virtual void print (int value, int base=DEC)=0

Println methods (with newline)

- virtual void println ()=0
- virtual void println (const __FlashStringHelper *str)=0
- virtual void println (const char *str)=0
- virtual void println (char c)=0
- virtual void println (uint8_t value, int base=DEC)=0
- virtual void println (uint16_t value, int base=DEC)=0
- virtual void println (uint32_t value, int base=DEC)=0
- virtual void println (int value, int base=DEC)=0

5.8.1 Detailed Description

Abstract interface for serial/debug output.
Provides a hardware-agnostic logging contract so that higher-level components (CryptnoxWallet, CW_SecureChannel) remain independent of the physical output device (UART, LCD, network, etc.).
On Arduino, the F() macro returns a __FlashStringHelper* so the dedicated overloads are called, keeping string literals in flash. On non-Arduino, F() is the identity macro (returns const char*), so the print(const char*) overload is called instead.
Definition at line 48 of file CW_Logger.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 ~CW_Logger()

```
virtual CW_Logger::~CW_Logger () [inline], [virtual]
```

Definition at line 80 of file CW_Logger.h.

5.8.3 Member Function Documentation

5.8.3.1 begin()

```
virtual bool CW_Logger::begin (
            unsigned long baudRate = 115200UL)  [pure virtual]
```

Initialize the logging interface.

Parameters

baudRate	Baud rate (relevant for UART implementations).

Returns

true if initialization succeeded, false otherwise.

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.

5.8.3.2 print() [1/7]

```
virtual void CW_Logger::print (
            char c)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.

5.8.3.3 print() [2/7]

```
virtual void CW_Logger::print (
            const __FlashStringHelper * str)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.

5.8.3.4 print() [3/7]

```
virtual void CW_Logger::print (
            const char * str)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.

5.8.3.5 print() [4/7]

```
virtual void CW_Logger::print (
            int value,
            int base = DEC)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.
References DEC.

5.8.3.6 print() [5/7]

```
virtual void CW_Logger::print (
            uint16_t value,
            int base = DEC)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.
References DEC.

5.8.3.7 print() [6/7]

```
virtual void CW_Logger::print (
            uint32_t value,
            int base = DEC)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.
References DEC.

5.8.3.8 print() [7/7]

```
virtual void CW_Logger::print (
            uint8_t value,
            int base = DEC)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.
References DEC.

5.8.3.9 println() [1/8]

```
virtual void CW_Logger::println ()  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.

5.8.3.10 println() [2/8]

```
virtual void CW_Logger::println (
            char c)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.

5.8.3.11 println() [3/8]

```
virtual void CW_Logger::println (
            const __FlashStringHelper * str)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.

5.8.3.12 println() [4/8]

```
virtual void CW_Logger::println (
            const char * str)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.

5.8.3.13 println() [5/8]

```
virtual void CW_Logger::println (
            int value,
            int base = DEC)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.
References DEC.

5.8.3.14 println() [6/8]

```
virtual void CW_Logger::println (
            uint16_t value,
            int base = DEC)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.
References DEC.

5.8.3.15 println() [7/8]

```
virtual void CW_Logger::println (
            uint32_t value,
            int base = DEC)  [pure virtual]
```

Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.
References DEC.

5.8.3.16 println() [8/8]

```
virtual void CW_Logger::println (
            uint8_t value,
            int base = DEC)  [pure virtual]
```

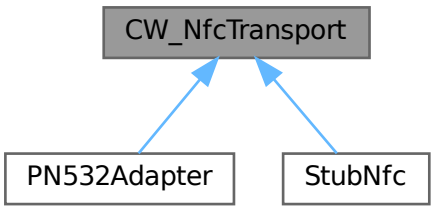
Implemented in ArduinoLoggerAdapter, NullLoggerAdapter, and StubLogger.

References DEC.
The documentation for this class was generated from the following file:

- src/cryptnox-sdk-cpp/CW_Logger.h

5.9 CW_NfcTransport Class Reference

Abstract interface for NFC transport operations.
`#include <CW_NfcTransport.h>`
Inheritance diagram for CW_NfcTransport:



Public Member Functions

- virtual bool begin ()=0
  Initialize the NFC transport hardware.
- virtual bool inListPassiveTarget ()=0
  Detect the presence of a passive ISO-DEP NFC target.
- virtual bool sendAPDU (const uint8_t *apdu, uint8_t apduLen, uint8_t *response, uint8_↩ t &responseLen)=0
  Send an APDU command to the card and receive the response.
- virtual bool sendAPDULarge (const uint8_t *apdu, uint8_t apduLen, uint8_t *response, uint16_t &responseLen)
  Send an APDU and receive a response that may exceed 255 bytes.
- virtual void resetReader ()=0
  Reset the NFC reader/field for the next card detection cycle.
- virtual bool printFirmwareVersion ()=0
  Print NFC module firmware version information to the logger.
- virtual ~CW_NfcTransport ()

5.9.1 Detailed Description

Abstract interface for NFC transport operations.
Defines the hardware-agnostic contract for NFC communication so that CW_SecureChannel and CryptnoxWallet remain independent of the physical NFC module (PN532, PN7150, etc.).
Definition at line 40 of file CW_NfcTransport.h.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 ~CW_NfcTransport()

`virtual CW_NfcTransport::~CW_NfcTransport () [inline], [virtual]`
Definition at line 103 of file CW_NfcTransport.h.

5.9.3 Member Function Documentation

5.9.3.1 begin()

```
virtual bool CW_NfcTransport::begin ()  [pure virtual]
```

Initialize the NFC transport hardware.

Returns

true if initialization succeeded, false otherwise.

Implemented in PN532Adapter, and StubNfc.

5.9.3.2 inListPassiveTarget()

```
virtual bool CW_NfcTransport::inListPassiveTarget ()  [pure virtual]
```

Detect the presence of a passive ISO-DEP NFC target.

Returns

true if a card is detected, false otherwise.

Implemented in PN532Adapter, and StubNfc.

5.9.3.3 printFirmwareVersion()

```
virtual bool CW_NfcTransport::printFirmwareVersion ()  [pure virtual]
```

Print NFC module firmware version information to the logger.

Returns

true if firmware info was retrieved successfully, false otherwise.

Implemented in PN532Adapter, and StubNfc.

5.9.3.4 resetReader()

```
virtual void CW_NfcTransport::resetReader ()  [pure virtual]
```

Reset the NFC reader/field for the next card detection cycle.
Implemented in PN532Adapter, and StubNfc.

5.9.3.5 sendAPDU()

```
virtual bool CW_NfcTransport::sendAPDU (
            const uint8_t * apdu,
            uint8_t apduLen,
            uint8_t * response,
            uint8_t & responseLen)  [pure virtual]
```

Send an APDU command to the card and receive the response.

Parameters

`in`	*apdu*	APDU command bytes.
`in`	*apduLen*	Length of the APDU command.
`out`	*response*	Buffer to receive the card response.
`out`	*responseLen*	Actual number of bytes written to `response`.

Returns

true if the exchange succeeded, false otherwise.

Implemented in PN532Adapter, and StubNfc.
Referenced by sendAPDULarge().

5.9.3.6 sendAPDULarge()

```
virtual bool CW_NfcTransport::sendAPDULarge (
            const uint8_t * apdu,
            uint8_t apduLen,
            uint8_t * response,
            uint16_t & responseLen)  [inline], [virtual]
```

Send an APDU and receive a response that may exceed 255 bytes.
Used for APDUs whose DataOut can be larger than a uint8_t can express (e.g. GET_↩ MANUFACTURER_CERTIFICATE returns up to 415 bytes). Implementations that cannot deliver more than 255 bytes may delegate to sendAPDU; the default below does exactly that.

Parameters

in	*apdu*	APDU command bytes.
in	*apduLen*	Length of the APDU command.
out	*response*	Buffer to receive the card response.
in,out	*responseLen*	On entry: capacity of response. On exit: actual bytes written.

Returns

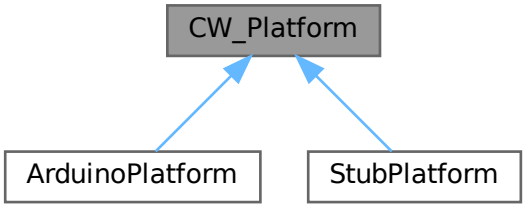
true if the exchange succeeded, false otherwise.

Reimplemented in PN532Adapter.
Definition at line 81 of file CW_NfcTransport.h.
References sendAPDU().
The documentation for this class was generated from the following file:

- src/cryptnox-sdk-cpp/CW_NfcTransport.h

5.10 CW_Platform Class Reference

Abstract interface for platform-specific operations used by the SDK.
`#include <CW_Platform.h>`
Inheritance diagram for CW_Platform:



Public Member Functions

- virtual void sleep_ms (uint32_t ms)=0

  Block for at least `ms` *milliseconds.*
- virtual ~CW_Platform ()

5.10.1 Detailed Description

Abstract interface for platform-specific operations used by the SDK.
Decouples the SDK core from any specific RTOS or bare-metal delay mechanism, allowing the same SDK to run on ESP32 (FreeRTOS), Arduino, and hosted (Linux/macOS) test environments.
Definition at line 39 of file CW_Platform.h.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 ~CW_Platform()

```
virtual CW_Platform::~CW_Platform () [inline], [virtual]
```
Definition at line 48 of file CW_Platform.h.

5.10.3 Member Function Documentation

5.10.3.1 sleep_ms()

```
virtual void CW_Platform::sleep_ms (
            uint32_t ms) [pure virtual]
```
Block for at least `ms` milliseconds.

Parameters

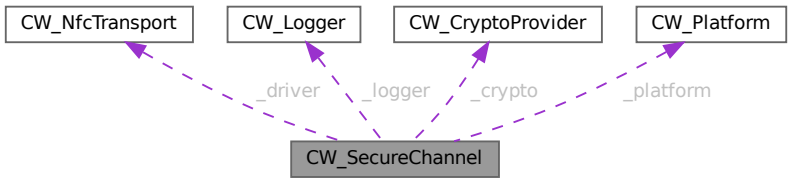
`in`	*ms*	Duration to sleep in milliseconds.

Implemented in ArduinoPlatform, and StubPlatform.
The documentation for this class was generated from the following file:

- src/cryptnox-sdk-cpp/CW_Platform.h

5.11 CW_SecureChannel Class Reference

Implements the Cryptnox secure channel protocol over NFC.
`#include <CW_SecureChannel.h>`
Collaboration diagram for CW_SecureChannel:



Public Member Functions

- CW_SecureChannel (CW_NfcTransport &driver, CW_Logger &logger, CW_CryptoProvider &crypto, CW_Platform &platform)

  Construct a CW_SecureChannel.
- CW_SecureChannel (const CW_SecureChannel &)=delete
- CW_SecureChannel & operator= (const CW_SecureChannel &)=delete
- bool begin ()

Initialize the NFC transport module.

- bool inListPassiveTarget ()

  Detect a passive NFC target (ISO-DEP card).
- void resetReader ()

  Reset the NFC reader hardware.
- bool printFirmwareVersion ()

  Print the NFC reader firmware version to the logger.
- bool selectApdu ()

  Send the SELECT APDU to activate the Cryptnox application.
- bool getCardCertificate (uint8_t *cardCertificate, uint8_t &cardCertificateLength)

  Retrieve the card's ephemeral public key via GET CARD CERTIFICATE.
- bool extractCardEphemeralKey (const uint8_t *cardCertificate, uint8_t *cardEphemeralPubKey, uint8_t *fullEphemeralPubKey65=NULL)

  Extract the card's ephemeral EC P-256 public key from a certificate.
- bool openSecureChannel (uint8_t *salt, uint8_t *clientPublicKey, uint8_t *clientPrivateKey, CW_Curve sessionCurve)

  Send OPEN SECURE CHANNEL and retrieve the session salt.
- bool mutuallyAuthenticate (CW_SecureSession &session, const uint8_t *salt, uint8_t *client↩ PublicKey, const uint8_t *clientPrivateKey, CW_Curve sessionCurve, const uint8_t *card↩ EphemeralPubKey)

  Perform ECDH derivation and MUTUALLY AUTHENTICATE with the card.
- bool getManufacturerCertificate (uint8_t *cert, uint16_t &certLen)

  Retrieve the manufacturer certificate stored in card flash.
- bool preFetchManufacturerCert ()

  Fetch and cache the manufacturer certificate before getCardCertificate().
- uint8_t verifyCertificateChain (const uint8_t *cardCert, uint8_t cardCertLen)

  Verify the full card certificate chain against the trusted CA.
- bool aesCbcEncrypt (CW_SecureSession &session, const uint8_t apdu[ ], uint16_t apdu↩ Length, const uint8_t data[ ], uint16_t dataLength, uint8_t *decryptedOutput=NULL, uint16_↩ t *decryptedOutputLength=NULL)

  AES-CBC encrypt + MAC, send APDU, and decrypt response.
- bool aesCbcDecrypt (const CW_SecureSession &session, uint8_t *response, size_t response↩ Len, uint8_t *macValue, uint8_t *decryptedOutput=NULL, uint16_t *decryptedOutput↩ Length=NULL)

  Verify MAC and decrypt an encrypted APDU response.
- bool checkStatusWord (const uint8_t *response, uint16_t responseLength, uint8_t sw1Expected, uint8_t sw2Expected)

  Verify the SW1/SW2 status word at the end of an APDU response.

Private Member Functions

- bool verifyEcdsaSha256 (const uint8_t *pubKey64, const uint8_t *message, uint16_t msgLen, const uint8_t *derSig, uint8_t derSigLen)

Static Private Member Functions

- static bool parseDerSigToRaw (const uint8_t *der, uint8_t derLen, uint8_t *raw64)

Private Attributes

- CW_NfcTransport & _driver

  NFC transport for APDU exchange.
- CW_Logger & _logger

  Logging interface.
- CW_CryptoProvider & _crypto

  Crypto operations (AES, SHA, ECDH, RNG).
- CW_Platform & _platform

  Platform abstraction (sleep_ms).
- uint8_t _lastNonce [CW_CERT_NONCE_SIZE]

  Nonce sent in the last getCardCertificate() call; checked in verifyCertificateChain().
- uint16_t _cachedMfCertLen

  Non-zero when s_mfCertBuf holds a valid pre-fetched manufacturer certificate.

5.11.1 Detailed Description

Implements the Cryptnox secure channel protocol over NFC.
Handles all low-level APDU exchanges required to establish and use a secure session with the Cryptnox smart card:

- Application selection (SELECT APDU)
- Card certificate retrieval and ephemeral key extraction
- ECDH-based session key derivation (OPEN SECURE CHANNEL + MUTUALLY AUTHENTICATE)
- AES-CBC-MAC secure messaging (encrypt / decrypt / MAC verify)
- Status word checking

CW_SecureChannel is composed inside CryptnoxWallet and is not intended to be used directly by application code.
Definition at line 64 of file CW_SecureChannel.h.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 CW_SecureChannel() [1/2]

```
CW_SecureChannel::CW_SecureChannel (
            CW_NfcTransport & driver,
            CW_Logger & logger,
            CW_CryptoProvider & crypto,
            CW_Platform & platform)
```

Construct a CW_SecureChannel.

Parameters

driver	Reference to the NFC transport.
logger	Reference to the logging interface.
crypto	Reference to the crypto provider.
platform	Reference to the platform abstraction (for sleep_ms).

Definition at line 87 of file CW_SecureChannel.cpp.
References _cachedMfCertLen, _crypto, _driver, _lastNonce, _logger, _platform, and platform.
Referenced by CW_SecureChannel(), and operator=().

5.11.2.2 CW_SecureChannel() [2/2]

```
CW_SecureChannel::CW_SecureChannel (
            const CW_SecureChannel & )  [delete]
```

References CW_SecureChannel().

5.11.3 Member Function Documentation

5.11.3.1 aesCbcDecrypt()

```
bool CW_SecureChannel::aesCbcDecrypt (
            const CW_SecureSession & session,
            uint8_t * response,
            size_t responseLen,
            uint8_t * macValue,
            uint8_t * decryptedOutput = NULL,
            uint16_t * decryptedOutputLength = NULL)
```

Verify MAC and decrypt an encrypted APDU response.
Internal helper called from aesCbcEncrypt — exposed for the fuzz harness. Verifies the response MAC against Kmac, then decrypts the payload with Kenc using the supplied IV (which is the MAC of the sent request, by protocol).

Parameters

in,out	*session*	Secure session.
in	*response*	Encrypted response buffer (MAC || cipher || SW).
in	*responseLen*	Response length.
in	*macValue*	MAC of the request — used as decrypt IV.
out	*decryptedOutput*	Optional plaintext output buffer.
out	*decryptedOutputLength*	Optional plaintext output length.

Returns

true if MAC matched and decryption succeeded, false otherwise.

Warning

A false return indicates either tampering or a corrupted channel — the session must not be reused without renegotiation.

Definition at line 625 of file CW_SecureChannel.cpp.
References _crypto, _logger, AES_BLOCK_SIZE, CW_SecureSession::aesKey, F, HEX, CW_SecureSession::macKey, s_apduBuf, s_dataBuf, s_macBuf, CW_Utils::safe_memcpy(), CW_Utils::secure_compare(), and CW_Utils::secure_wipe().
Referenced by aesCbcEncrypt().

5.11.3.2 aesCbcEncrypt()

```
bool CW_SecureChannel::aesCbcEncrypt (
            CW_SecureSession & session,
            const uint8_t apdu[],
            uint16_t apduLength,
            const uint8_t data[],
            uint16_t dataLength,
            uint8_t * decryptedOutput = NULL,
            uint16_t * decryptedOutputLength = NULL)
```

AES-CBC encrypt + MAC, send APDU, and decrypt response.

Performs one secure messaging round-trip: pads and encrypts `data` with Kenc using the current IV; computes a CMAC over (header || cipher) with Kmac; sends the wrapped APDU; on the response, verifies the MAC and decrypts the payload. The new IV for the next call is taken from the last cipher block (rolling IV).

Parameters

`in,out`	*session*	Secure session (Kenc / Kmac / IV).
`in`	*apdu*	APDU header (CLA, INS, P1, P2).
`in`	*apduLength*	Header length (must be 4).
`in`	*data*	Plaintext payload.
`in`	*dataLength*	Plaintext length ($\leq$ CW_USER_DATA_PAGE_SIZE).
`out`	*decryptedOutput*	Optional buffer for the decrypted response payload.
`out`	*decryptedOutputLength*	Optional pointer to receive the decrypted payload length.

Returns

true if the APDU was sent and the response MAC verified + decrypted successfully; false on bad parameters, transport failure, MAC mismatch, or unexpected status word.

Precondition

`session` must be the output of a successful mutuallyAuthenticate call.

Warning

Mutates `session.iv`. On any failure the IV may be left in an undefined state — treat the session as broken and tear it down with CW_SecureSession::clear.

Reuses module-private static buffers (`s_apduBuf`, `s_macBuf`, `s_dataBuf`). Not safe to call concurrently from multiple tasks; serialise at the application level.

One secure-messaging round-trip is built in five stages, reusing module- private scratch buffers (`s_↩apduBuf`, `s_macBuf`, `s_dataBuf`) to keep the call-site stack frame small:

1. Plaintext padding — ISO/IEC 9797-1 Method 2 (bit padding) is appended to `data` so the length is a multiple of the AES block size.
2. Encryption — AES-256-CBC under Kenc with the current rolling IV.
3. MAC computation — AES-CMAC over (APDU header || Lc || ciphertext) under Kmac. The MAC's last 8 bytes are prepended to the ciphertext in the outgoing APDU.
4. Transport — the assembled APDU goes to the card; the response is structured as (MAC[8] || cipher || SW1 SW2).
5. Response decryption — delegated to aesCbcDecrypt, which verifies the response MAC against Kmac (using the request MAC as the IV per protocol) before decrypting under Kenc.

IV update — on success, the last 16 bytes of the response ciphertext become the new session IV (rolling IV). On any failure path the IV is left in an undefined state, which is why the caller must treat a false return as a dead session.
Definition at line 501 of file CW_SecureChannel.cpp.
References _crypto, _driver, _logger, AES_BLOCK_SIZE, aesCbcDecrypt(), CW_SecureSession::aesKey, APDU_LC_LEN, checkStatusWord(), CW_SecureSession::clear(), CW_IV_SIZE, F, HEX, INPUT_BUFFER_LIMIT, CW_SecureSession::iv, MAC_APDU_LEN, CW_SecureSession::macKey, MAX_MAC_DATA_LEN, s_apduBuf, s_dataBuf, s_macBuf, CW_Utils::safe_memcpy(), CW_Utils::secure_wipe(), and SEND_APDU_MAX_LEN.

5.11.3.3 begin()

```
bool CW_SecureChannel::begin ()
```
Initialize the NFC transport module.

Returns

true if the module was successfully initialised, false otherwise.

Definition at line 100 of file CW_SecureChannel.cpp.
References _driver.

5.11.3.4 checkStatusWord()

```
bool CW_SecureChannel::checkStatusWord (
            const uint8_t * response,
            uint16_t responseLength,
            uint8_t sw1Expected,
            uint8_t sw2Expected)
```
Verify the SW1/SW2 status word at the end of an APDU response.

Parameters

response	APDU response buffer.
responseLength	Response length.
sw1Expected	Expected SW1 byte.
sw2Expected	Expected SW2 byte.

Returns

true if last two bytes match expectations, false otherwise.

Definition at line 120 of file CW_SecureChannel.cpp.
References _logger, F, and HEX.
Referenced by aesCbcEncrypt(), getCardCertificate(), getManufacturerCertificate(), mutuallyAuthenticate(), openSecureChannel(), and selectApdu().

5.11.3.5 extractCardEphemeralKey()

```
bool CW_SecureChannel::extractCardEphemeralKey (
            const uint8_t * cardCertificate,
            uint8_t * cardEphemeralPubKey,
            uint8_t * fullEphemeralPubKey65 = NULL)
```
Extract the card's ephemeral EC P-256 public key from a certificate.

Parameters

in	*cardCertificate*	Raw certificate bytes.
out	*cardEphemeralPubKey*	64-byte key (X‖Y, no 0x04 prefix) for ECDH.
out	*fullEphemeralPubKey65*	Optional 65-byte key including 0x04 prefix.

Returns

true on success, false otherwise.

Definition at line 232 of file CW_SecureChannel.cpp.

5.11.3.6 getCardCertificate()

```
bool CW_SecureChannel::getCardCertificate (
            uint8_t * cardCertificate,
            uint8_t & cardCertificateLength)
```

Retrieve the card's ephemeral public key via GET CARD CERTIFICATE.
Sends a random challenge nonce to the card and stores it internally. The nonce echo check is performed inside verifyCertificateChain() to ensure replay protection is coupled with signature verification.

Parameters

out	*cardCertificate*	Buffer to receive the raw certificate bytes.
out	*cardCertificateLength*	Actual certificate length (bytes).

Returns

true on success, false otherwise.

Definition at line 184 of file CW_SecureChannel.cpp.
References _crypto, _driver, _lastNonce, _logger, checkStatusWord(), F, GETCARDCERTIFICATE_IN_BYTES, RANDOM_BYTES, RESPONSE_GETCARDCERTIFICATE_IN_BYTES, RESPONSE_STATUS_WORDS_IN_BYTES, and CW_Utils::safe_memcpy().

5.11.3.7 getManufacturerCertificate()

```
bool CW_SecureChannel::getManufacturerCertificate (
            uint8_t * cert,
            uint16_t & certLen)
```

Retrieve the manufacturer certificate stored in card flash.

Parameters

out	*cert*	Buffer to receive the raw DER certificate bytes.
out	*certLen*	Actual certificate length written.

Returns

true on success, false if APDU fails or buffer too small.

Definition at line 1098 of file CW_SecureChannel.cpp.
References _driver, _logger, checkStatusWord(), CW_MANUF_CERT_MAX_BYTES, F, RESPONSE_GETMANUFACTURERCERT_PAGE_IN_BYTES, RESPONSE_STATUS_WORDS_IN_BYTES, and CW_Utils::safe_memcpy().
Referenced by preFetchManufacturerCert(), and verifyCertificateChain().

5.11.3.8 inListPassiveTarget()

```
bool CW_SecureChannel::inListPassiveTarget ()
```

Detect a passive NFC target (ISO-DEP card).

Returns

true if a card was found, false otherwise.

Definition at line 104 of file CW_SecureChannel.cpp.
References _driver.

5.11.3.9 mutuallyAuthenticate()

```
bool CW_SecureChannel::mutuallyAuthenticate (
            CW_SecureSession & session,
            const uint8_t * salt,
            uint8_t * clientPublicKey,
            const uint8_t * clientPrivateKey,
            CW_Curve sessionCurve,
            const uint8_t * cardEphemeralPubKey)
```

Perform ECDH derivation and MUTUALLY AUTHENTICATE with the card.
Final step of the secure channel handshake:

1. ECDH shared secret = clientPrivateKey cardEphemeralPubKey
2. (Kenc || Kmac || IV) ← SHA-512(salt || pairingKey || sharedSecret)
3. Encrypts a 16-byte random challenge with the new Kenc and sends it inside the MUTUALLY AUTHENTICATE APDU
4. Verifies the card returns the same plaintext when re-encrypting its own counter — this proves the card knows Kenc

Parameters

`out`	*session*	Secure session populated with derived keys + initial IV.
`in`	*salt*	32-byte salt from openSecureChannel.
`in`	*clientPublicKey*	64-byte client public key.
`in`	*clientPrivateKey*	32-byte client private key.
`in`	*sessionCurve*	ECC curve.
`in`	*cardEphemeralPubKey*	64-byte card ephemeral public key.

Returns

true on success, false if ECDH failed, the card's challenge response did not match, or any APDU exchange failed.

Precondition

openSecureChannel must have been called and returned true.

Postcondition

On true: `session` has Kenc, Kmac, and rolling IV ready for aesCbcEncrypt. On false: `session` is left untouched and must not be used.

Warning

All ephemeral key material in the caller's stack (`clientPrivateKey`, `salt`) must be wiped after this call.

Cryptographic flow:

1. Compute the ECDH shared secret S = ECDH(clientPrivateKey, cardEphemeralPubKey) on the negotiated curve.
2. Derive 80 bytes of keying material via SHA-512 over (salt || S || pairingDataHash) and split into:
   - Kenc[32]: AES-256 session encryption key
   - Kmac[32]: AES-256 session MAC key

- IV[16] : initial rolling IV

3. Send MUTUALLY AUTHENTICATE with a random 16-byte challenge encrypted under Kenc / IV. The card must answer with the same 16 bytes re-encrypted with the new IV — a verification that fails fast on any key-derivation mismatch.
4. On success, populate `session` and wipe `sharedSecret` from the stack.

Failure modes that cause an early-exit with a wiped session:

- ECDH returned zero or invalid (curve mismatch)
- APDU transport failure
- Card challenge response mismatch (active attacker or wrong card)

Definition at line 334 of file CW_SecureChannel.cpp.
References _crypto, _driver, _logger, AES_BLOCK_SIZE, CW_SecureSession::aesKey, APDU_HEADER_LEN, APDU_LC_LEN, checkStatusWord(), CW_SecureSession::clear(), COMMON_PAIRING_DATA, CW_AESKEY_SIZE, CW_IV_SIZE, CW_MACKEY_SIZE, F, CW_SecureSession::iv, CW_SecureSession::macKey, REQUEST_MUTUALLYAUTHENTICATE_IN_BYTES, RESPONSE_MUTUALLYAUTHENTICATE_IN_BYTES, CW_Utils::safe_memcpy(), and CW_Utils::secure_wipe().

5.11.3.10 openSecureChannel()

```
bool CW_SecureChannel::openSecureChannel (
            uint8_t * salt,
            uint8_t * clientPublicKey,
            uint8_t * clientPrivateKey,
            CW_Curve sessionCurve)
```

Send OPEN SECURE CHANNEL and retrieve the session salt.
Generates a client EC key pair via the crypto provider and sends the public key to the card; the card responds with a 32-byte salt that later feeds into Kenc / Kmac derivation in mutuallyAuthenticate.

Parameters

`out`	*salt*	32-byte session salt returned by the card.
`out`	*clientPublicKey*	64-byte freshly generated client public key.
`out`	*clientPrivateKey*	32-byte freshly generated client private key.
`in`	*sessionCurve*	ECC curve for key generation (secp256r1).

Returns

true on success, false otherwise.

Precondition

selectApdu must have been called successfully.

Warning

`clientPrivateKey` is sensitive — the caller MUST wipe it via CW_Utils::secure_wipe on every exit path.

Definition at line 265 of file CW_SecureChannel.cpp.
References _crypto, _driver, _logger, checkStatusWord(), CLIENT_PUBLIC_KEY_SIZE, F, OPENSECURECHANNEL_SALT_IN_BYTES, RESPONSE_OPENSECURECHANNEL_IN_BYTES, and CW_Utils::safe_memcpy().

5.11.3.11 operator=()

```
CW_SecureChannel & CW_SecureChannel::operator= (
            const CW_SecureChannel & )  [delete]
```

References CW_SecureChannel().

5.11.3.12 parseDerSigToRaw()

```
bool CW_SecureChannel::parseDerSigToRaw (
            const uint8_t * der,
            uint8_t derLen,
            uint8_t * raw64)  [static], [private]
```

Definition at line 1018 of file CW_SecureChannel.cpp.
References CW_Utils::safe_memcpy().
Referenced by DerFuzzTarget::parseDerSigToRaw(), and verifyEcdsaSha256().

5.11.3.13 preFetchManufacturerCert()

```
bool CW_SecureChannel::preFetchManufacturerCert ()
```

Fetch and cache the manufacturer certificate before getCardCertificate().
The Cryptnox card state machine advances after GET_CARD_CERTIFICATE (INS=F8) and will not respond to GET_MANUFACTURER_CERTIFICATE (INS=F7) after that point. Call this method immediately after selectApdu() and before getCardCertificate() so that verifyCertificateChain() can use the cached copy without an APDU.

Returns

true if the certificate was fetched and cached, false otherwise.

Definition at line 1194 of file CW_SecureChannel.cpp.
References _cachedMfCertLen, getManufacturerCertificate(), and s_mfCertBuf.

5.11.3.14 printFirmwareVersion()

```
bool CW_SecureChannel::printFirmwareVersion ()
```

Print the NFC reader firmware version to the logger.

Returns

true on success, false otherwise.

Definition at line 112 of file CW_SecureChannel.cpp.
References _driver.

5.11.3.15 resetReader()

```
void CW_SecureChannel::resetReader ()
```

Reset the NFC reader hardware.
Definition at line 108 of file CW_SecureChannel.cpp.
References _driver.

5.11.3.16 selectApdu()

```
bool CW_SecureChannel::selectApdu ()
```

Send the SELECT APDU to activate the Cryptnox application.

Returns

true on success, false otherwise.

Definition at line 155 of file CW_SecureChannel.cpp.
References _driver, _logger, checkStatusWord(), F, and RESPONSE_SELECT_IN_BYTES.

5.11.3.17 verifyCertificateChain()

```
uint8_t CW_SecureChannel::verifyCertificateChain (
            const uint8_t * cardCert,
            uint8_t cardCertLen)
```

Verify the full card certificate chain against the trusted CA.
Walks the cached manufacturer certificate (fetched earlier by preFetchManufacturerCert), verifies its ECDSA signature against each entry in CW_TRUSTED_CA_KEYS, then verifies the card's ephemeral certificate against the manufacturer public key. Also checks that the challenge nonce sent in getCardCertificate was echoed back inside the card certificate.

Parameters

in	*cardCert*	Raw card certificate bytes (typically 146 bytes).
in	*cardCertLen*	Length of `cardCert`.

Returns

One of the `CW_CERT_*` result codes:

Return values

CW_CERT_OK	Chain verified end-to-end.
CW_CERT_FORMAT_ERROR	Malformed certificate / unexpected TLV.
CW_CERT_NONCE_MISMATCH	Card did not echo the challenge nonce.
CW_CERT_CARD_SIG_INVALID	Card cert ECDSA signature failed verification.
CW_CERT_MANUF_SIG_INVALID	Manufacturer cert signature does not match any trusted CA key.
CW_CERT_KEY_NOT_FOUND	Device public-key OID not found in the certificate.

Precondition

preFetchManufacturerCert must have been called and returned true (the manufacturer certificate is cached internally and not re-fetchable after getCardCertificate).

getCardCertificate must have been called so the challenge nonce is recorded internally.

Two-step ECDSA chain walk against the pinned trusted CAs in CW_TRUSTED_CA_KEYS (currently a single secp256r1 key, CW_CA_DLT_PUBKEY):

1. Manufacturer certificate — the cached DER blob fetched by preFetchManufacturerCert is parsed to extract its EC public key and its ECDSA signature. The signature is verified against every entry in the trusted-CA table; the first match wins. The extracted manufacturer public key becomes the trusted issuer for step 2.
2. Card certificate — `cardCert` is parsed for the device's ephemeral public key, the manufacturer ECDSA signature over that key, and the echoed challenge nonce. The nonce is compared (constant-time) against the value stored by getCardCertificate; mismatch immediately returns CW_CERT_NONCE_MISMATCH to defeat replay. The signature is then verified against the manufacturer public key from step 1.

Both signatures are SHA-256-then-ECDSA over the relevant TBS bytes. Any TLV parsing error short-circuits with CW_CERT_FORMAT_ERROR rather than touching the verifier — DER parsing is the largest attack surface here and is independently fuzzed (see `fuzz/fuzz_der.cpp`).
Definition at line 1227 of file CW_SecureChannel.cpp.

References _cachedMfCertLen, _lastNonce, _logger, CW_CERT_CARD_SIG_INVALID, CW_CERT_FORMAT_ERROR, CW_CERT_KEY_NOT_FOUND, CW_CERT_MANUF_SIG_INVALID, CW_CERT_NONCE_MISMATCH, CW_CERT_NONCE_SIZE, CW_CERT_OK, CW_TRUSTED_CA_COUNT, CW_TRUSTED_CA_KEYS, derWalkMfCert(), F, getManufacturerCertificate(), s_mfCertBuf, CW_Utils::secure_compare(), CW_Utils::secure_wipe(), and verifyEcdsaSha256().

5.11.3.18 verifyEcdsaSha256()

```
bool CW_SecureChannel::verifyEcdsaSha256 (
            const uint8_t * pubKey64,
            const uint8_t * message,
            uint16_t msgLen,
            const uint8_t * derSig,
            uint8_t derSigLen)  [private]
```

Definition at line 1082 of file CW_SecureChannel.cpp.
References _crypto, CW_CURVE_SECP256R1, and parseDerSigToRaw().
Referenced by verifyCertificateChain().

5.11.4 Member Data Documentation

5.11.4.1 _cachedMfCertLen

```
uint16_t CW_SecureChannel::_cachedMfCertLen  [private]
```

Non-zero when s_mfCertBuf holds a valid pre-fetched manufacturer certificate.
Definition at line 322 of file CW_SecureChannel.h.
Referenced by CW_SecureChannel(), preFetchManufacturerCert(), and verifyCertificateChain().

5.11.4.2 _crypto

```
CW_CryptoProvider& CW_SecureChannel::_crypto  [private]
```

Crypto operations (AES, SHA, ECDH, RNG).
Definition at line 315 of file CW_SecureChannel.h.
Referenced by aesCbcDecrypt(), aesCbcEncrypt(), CW_SecureChannel(), getCardCertificate(), mutuallyAuthenticate(), openSecureChannel(), and verifyEcdsaSha256().

5.11.4.3 _driver

```
CW_NfcTransport& CW_SecureChannel::_driver  [private]
```

NFC transport for APDU exchange.
Definition at line 313 of file CW_SecureChannel.h.
Referenced by aesCbcEncrypt(), begin(), CW_SecureChannel(), getCardCertificate(), getManufacturerCertificate(), inListPassiveTarget(), mutuallyAuthenticate(), openSecureChannel(), printFirmwareVersion(), resetReader(), and selectApdu().

5.11.4.4 _lastNonce

```
uint8_t CW_SecureChannel::_lastNonce[CW_CERT_NONCE_SIZE]  [private]
```

Nonce sent in the last getCardCertificate() call; checked in verifyCertificateChain().
Definition at line 319 of file CW_SecureChannel.h.
Referenced by CW_SecureChannel(), getCardCertificate(), and verifyCertificateChain().

5.11.4.5 _logger

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

Logging interface.
Definition at line 314 of file CW_SecureChannel.h.
Referenced by aesCbcDecrypt(), aesCbcEncrypt(), checkStatusWord(), CW_SecureChannel(), getCardCertificate(), getManufacturerCertificate(), mutuallyAuthenticate(), openSecureChannel(), selectApdu(), and verifyCertificateChain().

5.11.4.6 _platform

```
CW_Platform& CW_SecureChannel::_platform  [private]
```
Platform abstraction (sleep_ms).
Definition at line 316 of file CW_SecureChannel.h.
Referenced by CW_SecureChannel().
The documentation for this class was generated from the following files:

- src/cryptnox-sdk-cpp/CW_SecureChannel.h
- src/cryptnox-sdk-cpp/CW_SecureChannel.cpp

5.12 CW_SecureSession Struct Reference

Holds cryptographic session state for reentrant secure channel operations.
`#include <CW_Defs.h>`

Public Member Functions

- CW_SecureSession ()
  Zero-initialise all session keys and IV.
- void clear ()
  Securely clear all session keys and IV.

Public Attributes

- uint8_t aesKey [CW_AESKEY_SIZE]
- uint8_t macKey [CW_MACKEY_SIZE]
- uint8_t iv [CW_IV_SIZE]

5.12.1 Detailed Description

Holds cryptographic session state for reentrant secure channel operations.
Encapsulates all session-specific cryptographic material (Kenc, Kmac, rolling IV), allowing functions to be reentrant by passing session state as a parameter.

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 168 of file CW_Defs.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 CW_SecureSession()

```
CW_SecureSession::CW_SecureSession ()  [inline]
```
Zero-initialise all session keys and IV.
Definition at line 174 of file CW_Defs.h.
References aesKey, iv, and macKey.

5.12.3 Member Function Documentation

5.12.3.1 clear()

```
void CW_SecureSession::clear ()  [inline]
```
Securely clear all session keys and IV.
Definition at line 181 of file CW_Defs.h.
References aesKey, iv, macKey, and CW_Utils::secure_wipe().
Referenced by CW_SecureChannel::aesCbcEncrypt(), CryptnoxWallet::connect(), CryptnoxWallet::disconnect(), and CW_SecureChannel::mutuallyAuthenticate().

5.12.4 Member Data Documentation

5.12.4.1 aesKey

`uint8_t CW_SecureSession::aesKey[CW_AESKEY_SIZE]`
AES-256 session encryption key (Kenc)
Definition at line 169 of file CW_Defs.h.
Referenced by CW_SecureChannel::aesCbcDecrypt(), CW_SecureChannel::aesCbcEncrypt(), clear(), CW_SecureSession(), CryptnoxWallet::isSecureChannelOpen(), and CW_SecureChannel::mutuallyAuthenticate().

5.12.4.2 iv

`uint8_t CW_SecureSession::iv[CW_IV_SIZE]`
Current AES-CBC IV (rolling IV)
Definition at line 171 of file CW_Defs.h.
Referenced by CW_SecureChannel::aesCbcEncrypt(), clear(), CW_SecureSession(), and CW_SecureChannel::mutuallyAuthenticate().

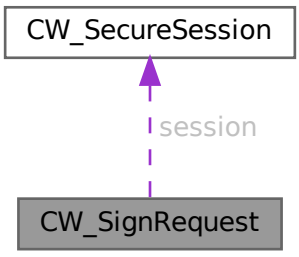
5.12.4.3 macKey

`uint8_t CW_SecureSession::macKey[CW_MACKEY_SIZE]`
AES-256 session MAC key (Kmac)
Definition at line 170 of file CW_Defs.h.
Referenced by CW_SecureChannel::aesCbcDecrypt(), CW_SecureChannel::aesCbcEncrypt(), clear(), CW_SecureSession(), and CW_SecureChannel::mutuallyAuthenticate().
The documentation for this struct was generated from the following file:

- src/cryptnox-sdk-cpp/CW_Defs.h

5.13 CW_SignRequest Struct Reference

Request parameters for CryptnoxWallet::sign.
`#include <CryptnoxWallet.h>`
Collaboration diagram for CW_SignRequest:



Public Member Functions

- CW_SignRequest (CW_SecureSession &sess, uint8_t kType=CW_SIGN_CURR_K1, uint8_t sigType=CW_SIGN_SIG_ECDSA_LOW_S, bool pinless=CW_SIGN_WITH_PIN)
  Construct a sign request with sensible defaults.
- ~CW_SignRequest ()
  Securely wipes the PIN buffer.

Public Attributes

- CW_SecureSession & session
- uint8_t keyType
- uint8_t signatureType
- uint8_t pin [CW_MAX_PIN_LENGTH]
- bool pinLessMode
- const uint8_t * hash
- uint8_t hashLength
- const uint8_t * derivePath
- uint8_t derivePathLength

5.13.1 Detailed Description

Request parameters for CryptnoxWallet::sign.
Owns the PIN buffer for the lifetime of the request — the destructor securely wipes it (CW_Utils::secure_wipe), so allocating the request on the stack inside a tight scope is the recommended pattern.

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 74 of file CryptnoxWallet.h.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 CW_SignRequest()

```
CW_SignRequest::CW_SignRequest (
            CW_SecureSession & sess,
            uint8_t kType = CW_SIGN_CURR_K1,
            uint8_t sigType = CW_SIGN_SIG_ECDSA_LOW_S,
            bool pinless = CW_SIGN_WITH_PIN)  [inline], [explicit]
```

Construct a sign request with sensible defaults.

Parameters

in	*sess*	Open secure session.
in	*kType*	Key type. Defaults to CW_SIGN_CURR_K1.
in	*sigType*	Signature type. Defaults to CW_SIGN_SIG_ECDSA_LOW_S.
in	*pinless*	PIN mode. Defaults to PIN required (CW_SIGN_WITH_PIN).

Definition at line 92 of file CryptnoxWallet.h.
References CW_SIGN_CURR_K1, CW_SIGN_SIG_ECDSA_LOW_S, CW_SIGN_WITH_PIN, derivePath, derivePathLength, hash, hashLength, keyType, pin, pinLessMode, session, and signatureType.

5.13.2.2 ~CW_SignRequest()

```
CW_SignRequest::~CW_SignRequest ()  [inline]
```

Securely wipes the PIN buffer.
Definition at line 103 of file CryptnoxWallet.h.
References pin, and CW_Utils::secure_wipe().

5.13.3 Member Data Documentation

5.13.3.1 derivePath

`const uint8_t* CW_SignRequest::derivePath`
BIP32 path bytes for DERIVE modes; `NULL` for CURR / PINLESS modes.
Definition at line 82 of file CryptnoxWallet.h.
Referenced by CryptnoxWallet::buildSignPayload(), CW_SignRequest(), and setup().

5.13.3.2 derivePathLength

`uint8_t CW_SignRequest::derivePathLength`
Length of `derivePath` in bytes (must be a multiple of 4).
Definition at line 83 of file CryptnoxWallet.h.
Referenced by CryptnoxWallet::buildSignPayload(), CW_SignRequest(), and setup().

5.13.3.3 hash

`const uint8_t* CW_SignRequest::hash`
Pointer to the hash to sign (typically 32 bytes — SHA-256 of the transaction).
Definition at line 80 of file CryptnoxWallet.h.
Referenced by CryptnoxWallet::buildSignPayload(), CW_SignRequest(), loop(), setup(), and CryptnoxWallet::validateSignRequest().

5.13.3.4 hashLength

`uint8_t CW_SignRequest::hashLength`
Length of `hash` in bytes (must be $\leq$ CW_HASH_SIZE).
Definition at line 81 of file CryptnoxWallet.h.
Referenced by CryptnoxWallet::buildSignPayload(), CW_SignRequest(), loop(), setup(), and CryptnoxWallet::validateSignRequest().

5.13.3.5 keyType

`uint8_t CW_SignRequest::keyType`
Key / path type — one of the `CW_SIGN_CURR_*`, `CW_SIGN_DERIVE_*`, `CW_SIGN_PINLESS_K1` constants.
Definition at line 76 of file CryptnoxWallet.h.
Referenced by CryptnoxWallet::buildSignPayload(), CW_SignRequest(), CryptnoxWallet::sendSignApdu(), and CryptnoxWallet::validateSignRequest().

5.13.3.6 pin

`uint8_t CW_SignRequest::pin[CW_MAX_PIN_LENGTH]`
PIN bytes (4–9 ASCII digits). Zero-padded; cleared in the destructor.
Definition at line 78 of file CryptnoxWallet.h.
Referenced by CryptnoxWallet::buildSignPayload(), CW_SignRequest(), loop(), setup(), CryptnoxWallet::validateSignRe and ~CW_SignRequest().

5.13.3.7 pinLessMode

`bool CW_SignRequest::pinLessMode`
false = PIN path, true = PIN-less path (requires `keyType` == CW_SIGN_PINLESS_K1).
Definition at line 79 of file CryptnoxWallet.h.
Referenced by CryptnoxWallet::buildSignPayload(), CW_SignRequest(), and CryptnoxWallet::validateSignRequest().

5.13.3.8 session

`CW_SecureSession& CW_SignRequest::session`
Reference to an open secure session.

Definition at line 75 of file CryptnoxWallet.h.
Referenced by CW_SignRequest(), CryptnoxWallet::sendSignApdu(), and CryptnoxWallet::validateSignRequest().

5.13.3.9 signatureType

`uint8_t CW_SignRequest::signatureType`
Signature format — one of `CW_SIGN_SIG_ECDSA_LOW_S`, `CW_SIGN_SIG_ECDSA_EOSIO`, `CW_SIGN_SIG_SCHNORR_BIP340`.
Definition at line 77 of file CryptnoxWallet.h.
Referenced by CW_SignRequest(), and CryptnoxWallet::sendSignApdu().
The documentation for this struct was generated from the following file:

- src/cryptnox-sdk-cpp/CryptnoxWallet.h

5.14 CW_SignResult Struct Reference

Result of CryptnoxWallet::sign.
`#include <CryptnoxWallet.h>`

Public Member Functions

- CW_SignResult ()
  Construct a default-failure result.

Public Attributes

- uint8_t signature [CW_RAW_SIGNATURE_SIZE]
- uint8_t errorCode

5.14.1 Detailed Description

Result of CryptnoxWallet::sign.
The error code is checked first: when it is CW_OK the signature is valid raw (r || s) on secp256k1 / secp256r1 (depending on the `keyType` used). On any other code `signature` is left zero.

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 116 of file CryptnoxWallet.h.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 CW_SignResult()

`CW_SignResult::CW_SignResult () [inline]`
Construct a default-failure result.
Definition at line 121 of file CryptnoxWallet.h.
References CW_NOK, errorCode, and signature.

5.14.3 Member Data Documentation

5.14.3.1 errorCode

`uint8_t CW_SignResult::errorCode`
CW_OK on success, otherwise a `CW_SIGN_*` / `CW_INVALID_SESSION` code.

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 118 of file CryptnoxWallet.h.
Referenced by CW_SignResult(), CryptnoxWallet::extractRawSignature(), loop(), CryptnoxWallet::sendSignApdu(), setup(), CryptnoxWallet::sign(), and CryptnoxWallet::validateSignRequest().

5.14.3.2 signature

```
uint8_t CW_SignResult::signature[CW_RAW_SIGNATURE_SIZE]
```
Raw 64-byte signature: r[32] || s[32]. Zero on failure.

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 117 of file CryptnoxWallet.h.
Referenced by CW_SignResult(), CryptnoxWallet::extractRawSignature(), loop(), setup(), and CryptnoxWallet::sign().
The documentation for this struct was generated from the following file:

- src/cryptnox-sdk-cpp/CryptnoxWallet.h

5.15 CW_Utils Class Reference

Portable utility functions for cryptographic and security operations.
`#include <CW_Utils.h>`

Static Public Member Functions

- static bool secure_compare (const uint8_t *a, const uint8_t *b, size_t len)
  Constant-time buffer comparison, resistant to timing side-channel attacks.
- static void secure_wipe (uint8_t *buf, size_t len)
  Securely zero a buffer, guaranteed not to be optimised away.
- static bool safe_memcpy (uint8_t *dst, size_t dstSize, const uint8_t *src, size_t count)
  Safe memcpy — validates pointers, sizes, and checks for overlap.
- static bool fill_secure_random (uint8_t *dest, size_t len)
  Fill `len` bytes at `dest` with cryptographically random data.

5.15.1 Detailed Description

Portable utility functions for cryptographic and security operations.
All methods here are platform-independent pure C++ with no dependency on Arduino or any hardware-specific library.
Hardware-specific helpers (e.g. TRNG byte generation) live in the concrete crypto provider implementation (ArduinoCryptoProvider).
Definition at line 45 of file CW_Utils.h.

5.15.2 Member Function Documentation

5.15.2.1 fill_secure_random()

```
bool CW_Utils::fill_secure_random (
            uint8_t * dest,
            size_t len)  [static]
```
Fill `len` bytes at `dest` with cryptographically random data.
The implementation is platform-specific. On ESP32 it calls esp_fill_random() after verifying that Wi-Fi or Bluetooth is active, ensuring the hardware TRNG is properly seeded. Returns false (hard failure) if neither radio is on (SEC-001).

Parameters

dest	Destination buffer.
len	Number of random bytes to generate.

Returns

true on success, false if dest is NULL, len is zero, or no radio is active.

Definition at line 109 of file fuzz_der.cpp.

5.15.2.2 safe_memcpy()

```
bool CW_Utils::safe_memcpy (
            uint8_t * dst,
            size_t dstSize,
            const uint8_t * src,
            size_t count)  [static]
```

Safe memcpy — validates pointers, sizes, and checks for overlap.
Safe memcpy — checks src, dst and size before copying.

Parameters

dst	Destination buffer.
dstSize	Capacity of the destination buffer.
src	Source buffer.
count	Number of bytes to copy.

Returns

true if copy succeeded, false if parameters are invalid.

true if copy succeeded, false otherwise.

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 50 of file CW_Utils.cpp.
Referenced by CW_SecureChannel::aesCbcDecrypt(), CW_SecureChannel::aesCbcEncrypt(), CryptnoxWallet::buildSignPayload(), CryptnoxWallet::extractRawSignature(), CW_SecureChannel::getCardCertificate(), CryptnoxWallet::getCardInfo(), CW_SecureChannel::getManufacturerCertificate(), loop(), CW_SecureChannel::mutuallyAuthenticate(), CW_SecureChannel::openSecureChannel(), CryptnoxWallet::parseDerSignature(), CW_SecureChannel::parseDerSigToRaw(), rlpFinalize(), setup(), and CryptnoxWallet::verifyPin().

5.15.2.3 secure_compare()

```
bool CW_Utils::secure_compare (
            const uint8_t * a,
            const uint8_t * b,
            size_t len)  [static]
```

Constant-time buffer comparison, resistant to timing side-channel attacks.
Always iterates over the full length regardless of where the first difference occurs, preventing an attacker from inferring the correct value byte-by-byte via timing measurements.

Parameters

a	Pointer to the first buffer.
b	Pointer to the second buffer.
len	Number of bytes to compare.

Returns

true if the buffers are identical, false otherwise.

Definition at line 22 of file CW_Utils.cpp.
Referenced by CW_SecureChannel::aesCbcDecrypt(), and CW_SecureChannel::verifyCertificateChain().

5.15.2.4 secure_wipe()

```
void CW_Utils::secure_wipe (
            uint8_t * buf,
            size_t len)  [static]
```

Securely zero a buffer, guaranteed not to be optimised away.
Uses a volatile pointer so the compiler cannot elide the writes, ensuring sensitive material is actually erased from memory.

Parameters

buf	Pointer to the buffer to wipe.
len	Number of bytes to zero.

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 37 of file CW_Utils.cpp.
Referenced by CW_SecureChannel::aesCbcDecrypt(), CW_SecureChannel::aesCbcEncrypt(), CW_SecureSession::clear(), encodeERC20Transfer(), CryptnoxWallet::establishSecureChannel(), CryptnoxWallet::extractRawSignature(), CryptnoxWallet::getCardInfo(), loop(), CW_SecureChannel::mutuallyAuthentica rlpEncodeSignedTx(), setup(), CryptnoxWallet::sign(), CW_SecureChannel::verifyCertificateChain(), CryptnoxWallet::verifyPin(), and CW_SignRequest::~CW_SignRequest().
The documentation for this class was generated from the following files:

- src/cryptnox-sdk-cpp/CW_Utils.h
- src/cryptnox-sdk-cpp/CW_Utils.cpp
- src/cryptnox-sdk-cpp/fuzz/fuzz_der.cpp

5.16 DerFuzzTarget Struct Reference

Static Public Member Functions

- static bool parseDerSigToRaw (const uint8_t *der, uint8_t derLen, uint8_t *raw64)

5.16.1 Detailed Description

Definition at line 121 of file fuzz_der.cpp.

5.16.2 Member Function Documentation

5.16.2.1 parseDerSigToRaw()

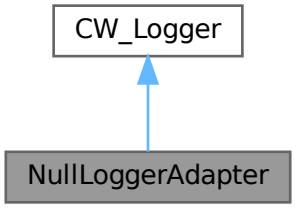
```
bool DerFuzzTarget::parseDerSigToRaw (
            const uint8_t * der,
            uint8_t derLen,
            uint8_t * raw64)  [inline], [static]
```

Definition at line 122 of file fuzz_der.cpp.
References CW_SecureChannel::parseDerSigToRaw().
Referenced by LLVMFuzzerTestOneInput().
The documentation for this struct was generated from the following file:

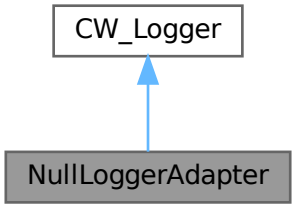
- src/cryptnox-sdk-cpp/fuzz/fuzz_der.cpp

5.17 NullLoggerAdapter Class Reference

No-op `CW_Logger` — guarantees nothing reaches the serial port.
`#include <NullLoggerAdapter.h>`
Inheritance diagram for NullLoggerAdapter:



Collaboration diagram for NullLoggerAdapter:



Public Member Functions

- NullLoggerAdapter ()=default
- ~NullLoggerAdapter () override=default
- NullLoggerAdapter (const NullLoggerAdapter &)=delete
- NullLoggerAdapter & operator= (const NullLoggerAdapter &)=delete

- bool begin (unsigned long=115200UL) override

  No-op begin(); returns true to match the interface contract.
- void print (const __FlashStringHelper *) override

  No-op.
- void print (const char *) override

  No-op.
- void print (char) override

  No-op.
- void print (uint8_t, int=DEC) override

  No-op.
- void print (uint16_t, int=DEC) override

  No-op.
- void print (uint32_t, int=DEC) override

  No-op.
- void print (int, int=DEC) override

  No-op.
- void println () override

  No-op.
- void println (const __FlashStringHelper *) override

  No-op.
- void println (const char *) override

  No-op.
- void println (char) override

  No-op.
- void println (uint8_t, int=DEC) override

  No-op.
- void println (uint16_t, int=DEC) override

  No-op.
- void println (uint32_t, int=DEC) override

  No-op.
- void println (int, int=DEC) override

  No-op.

Public Member Functions inherited from CW_Logger

- virtual ~CW_Logger ()

5.17.1 Detailed Description

No-op `CW_Logger` — guarantees nothing reaches the serial port.
Every `CW_Logger` method is an empty inline override, so:

- The compiler dead-code-eliminates the formatting work at the call site (no string format, no hex conversion, no `Serial` write).
- Even with `CW_DEBUG_LOGGING` enabled in the build, no APDU contents, no session-key fragments, no PIN-handling traces leave the chip through the UART (audit findings LOW-03 / MED-02).
- Required for any deployment where a physically accessible UART must not become a side-channel.

Example — production wiring

```
NullLoggerAdapter     logger;             // silent
ArduinoCryptoProvider crypto;
PN532Adapter          nfc(logger, 10);
CryptnoxWallet        wallet(nfc, logger, crypto, platform);
```

Note

Drop-in interchangeable with ArduinoLoggerAdapter — flip the one declaration line to switch between dev and production builds.

Definition at line 50 of file NullLoggerAdapter.h.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 NullLoggerAdapter() [1/2]

```
NullLoggerAdapter::NullLoggerAdapter ()  [default]
```

Referenced by NullLoggerAdapter(), and operator=().

5.17.2.2 ~NullLoggerAdapter()

```
NullLoggerAdapter::~NullLoggerAdapter ()  [override], [default]
```

5.17.2.3 NullLoggerAdapter() [2/2]

```
NullLoggerAdapter::NullLoggerAdapter (
            const NullLoggerAdapter & )  [delete]
```

References NullLoggerAdapter().

5.17.3 Member Function Documentation

5.17.3.1 begin()

```
bool NullLoggerAdapter::begin (
            unsigned long  = 115200UL)  [inline], [override], [virtual]
```

No-op begin(); returns true to match the interface contract.
Implements CW_Logger.
Definition at line 59 of file NullLoggerAdapter.h.

5.17.3.2 operator=()

```
NullLoggerAdapter & NullLoggerAdapter::operator= (
            const NullLoggerAdapter & )  [delete]
```

References NullLoggerAdapter().

5.17.3.3 print() [1/7]

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

No-op.
Implements CW_Logger.
Definition at line 63 of file NullLoggerAdapter.h.

5.17.3.4 print() [2/7]

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

No-op.
Implements CW_Logger.
Definition at line 61 of file NullLoggerAdapter.h.

5.17.3.5 print() [3/7]

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

No-op.
Implements CW_Logger.
Definition at line 62 of file NullLoggerAdapter.h.

5.17.3.6 print() [4/7]

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

No-op.
Implements CW_Logger.
Definition at line 67 of file NullLoggerAdapter.h.
References DEC.

5.17.3.7 print() [5/7]

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

No-op.
Implements CW_Logger.
Definition at line 65 of file NullLoggerAdapter.h.
References DEC.

5.17.3.8 print() [6/7]

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

No-op.
Implements CW_Logger.
Definition at line 66 of file NullLoggerAdapter.h.
References DEC.

5.17.3.9 print() [7/7]

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

No-op.
Implements CW_Logger.
Definition at line 64 of file NullLoggerAdapter.h.
References DEC.

5.17.3.10 println() [1/8]

```
void NullLoggerAdapter::println ()  [inline], [override], [virtual]
```

No-op.
Implements CW_Logger.
Definition at line 69 of file NullLoggerAdapter.h.

5.17.3.11 println() [2/8]

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

No-op.

Implements CW_Logger.
Definition at line 72 of file NullLoggerAdapter.h.

5.17.3.12 println() [3/8]

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

No-op.
Implements CW_Logger.
Definition at line 70 of file NullLoggerAdapter.h.

5.17.3.13 println() [4/8]

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

No-op.
Implements CW_Logger.
Definition at line 71 of file NullLoggerAdapter.h.

5.17.3.14 println() [5/8]

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

No-op.
Implements CW_Logger.
Definition at line 76 of file NullLoggerAdapter.h.
References DEC.

5.17.3.15 println() [6/8]

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

No-op.
Implements CW_Logger.
Definition at line 74 of file NullLoggerAdapter.h.
References DEC.

5.17.3.16 println() [7/8]

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

No-op.
Implements CW_Logger.
Definition at line 75 of file NullLoggerAdapter.h.
References DEC.

5.17.3.17 println() [8/8]

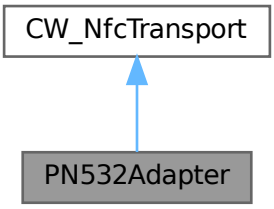
```
void NullLoggerAdapter::println (
            uint8_t ,
            int  = DEC)  [inline], [override], [virtual]
```

No-op.
Implements CW_Logger.
Definition at line 73 of file NullLoggerAdapter.h.
References DEC.
The documentation for this class was generated from the following file:

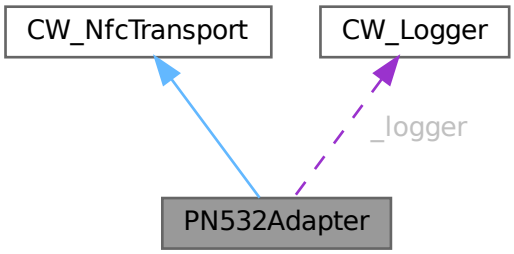
- src/NullLoggerAdapter.h

5.18 PN532Adapter Class Reference

`CW_NfcTransport` implementation over the Adafruit_PN532 driver.
`#include <PN532Adapter.h>`
Inheritance diagram for PN532Adapter:



Collaboration diagram for PN532Adapter:



Public Member Functions

- PN532Adapter (CW_Logger &logger, uint8_t ssPin, SPIClass *theSPI=&SPI)
  *Construct an adapter wired over **hardware SPI**.*
- PN532Adapter (CW_Logger &logger, uint8_t clk, uint8_t miso, uint8_t mosi, uint8_t ss)
  *Construct an adapter wired over **bit-banged software SPI**.*
- PN532Adapter (CW_Logger &logger, uint8_t irqPin, uint8_t resetPin, TwoWire *wire=&Wire)
  *Construct an adapter wired over **I2C**.*
- PN532Adapter (CW_Logger &logger, uint8_t resetPin, HardwareSerial *uartSerial)
  *Construct an adapter wired over **UART**.*
- ~PN532Adapter () override
  Release the heap-allocated `Adafruit_PN532` instance.
- PN532Adapter (const PN532Adapter &)=delete
- PN532Adapter & operator= (const PN532Adapter &)=delete

CW_NfcTransport interface

- bool begin () override

Initialise the PN532 and probe its firmware version.

- bool sendAPDU (const uint8_t *apdu, uint8_t apduLen, uint8_t *response, uint8_t &responseLen) override
  Exchange one ISO-DEP APDU with the currently selected card.
- bool sendAPDULarge (const uint8_t *apdu, uint8_t apduLen, uint8_t *response, uint16_t &responseLen) override
  APDU exchange that can return more than 255 bytes.
- bool inListPassiveTarget () override
  Poll for a card in the field.
- void resetReader () override
  Reset the PN532 into a clean idle state.
- bool printFirmwareVersion () override
  Pretty-print the PN532 firmware version + supported features.

Public Member Functions inherited from CW_NfcTransport

- virtual ~CW_NfcTransport ()

Private Attributes

- CW_Logger * _logger
  Logger reference (non-owning).
- PN532Interface _interface
  Wiring variant the active constructor selected.
- Adafruit_PN532 * _nfc
  Owned Adafruit_PN532 driver instance.

5.18.1 Detailed Description

`CW_NfcTransport` implementation over the Adafruit_PN532 driver.
Concrete transport the host integration constructs once per reader and passes by reference to CryptnoxWallet. The class owns the underlying `Adafruit_PN532` instance (heap-allocated by the constructor matching the wiring, released by the destructor) — callers do not need to manage its lifetime.
Forwards each CW_NfcTransport call to the equivalent Adafruit_PN532 method, plus a bounds check on APDU length (HIGH-05) and an optional hex-dump of the response when `CW_DEBUG_LOGGING` is set.

Example — SPI hardware (Arduino UNO R4 default SPI pins)

```
ArduinoLoggerAdapter logger;
PN532Adapter         nfc(logger, 10);           // SS on D10
nfc.begin();
```

Example — I2C (PN532 with IRQ + RESET)

```
PN532Adapter nfc(logger, 2, 3);                 // IRQ=D2, RESET=D3
```

Note

Non-copyable: the wrapped Adafruit_PN532 is owned uniquely.

Warning

The PN532 module has a hard 255-byte limit on ISO-DEP APDUs. sendAPDU rejects anything longer (HIGH-05) — chunking must be done in the protocol layer.

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 86 of file PN532Adapter.h.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 PN532Adapter() [1/5]

```
PN532Adapter::PN532Adapter (
            CW_Logger & logger,
            uint8_t ssPin,
            SPIClass * theSPI = &SPI)
```

Construct an adapter wired over **hardware SPI**.
Uses the MCU's SPI peripheral; SCK/MOSI/MISO are fixed by the SPI block (D13/D11/D12 on the UNO R4). Only the slave-select pin is configurable.

Parameters

in	*logger*	Logger sink for debug and error output.
in	*ssPin*	GPIO connected to the PN532 SS / NSS pin.
in	*theSPI*	Optional SPI instance (defaults to the global `SPI`).

Definition at line 22 of file PN532Adapter.cpp.
References _interface, _logger, and _nfc.
Referenced by operator=(), and PN532Adapter().

5.18.2.2 PN532Adapter() [2/5]

```
PN532Adapter::PN532Adapter (
            CW_Logger & logger,
            uint8_t clk,
            uint8_t miso,
            uint8_t mosi,
            uint8_t ss)
```

Construct an adapter wired over **bit-banged software SPI**.
Use this when the hardware SPI peripheral is already taken by another device. Any four GPIOs work; throughput is significantly lower than hardware SPI.

Parameters

in	*logger*	Logger sink.
in	*clk*	GPIO for SPI clock.
in	*miso*	GPIO for MISO (PN532 → MCU).
in	*mosi*	GPIO for MOSI (MCU → PN532).
in	*ss*	GPIO for slave-select.

Definition at line 27 of file PN532Adapter.cpp.
References _interface, _logger, and _nfc.

5.18.2.3 PN532Adapter() [3/5]

```
PN532Adapter::PN532Adapter (
            CW_Logger & logger,
            uint8_t irqPin,
            uint8_t resetPin,
            TwoWire * wire = &Wire)
```

Construct an adapter wired over **I2C**.
Requires the PN532 SEL0/SEL1 jumpers to be set to I2C and external pull-ups on SDA/SCL (the PN532 breakouts from Adafruit include them).

Parameters

in	*logger*	Logger sink.
in	*irqPin*	GPIO connected to the PN532 IRQ line.
in	*resetPin*	GPIO connected to the PN532 RSTPDN line.
in	*wire*	Optional `TwoWire` instance (defaults to the global `Wire`).

Definition at line 32 of file PN532Adapter.cpp.
References _interface, _logger, and _nfc.

5.18.2.4 PN532Adapter() [4/5]

```
PN532Adapter::PN532Adapter (
            CW_Logger & logger,
            uint8_t resetPin,
            HardwareSerial * uartSerial)
```

Construct an adapter wired over **UART**.
The PN532 must be configured at 115200 baud by its SEL jumpers. UART is the slowest mode but tolerates long cable runs better than SPI/I2C.

Parameters

in	*logger*	Logger sink.
in	*resetPin*	GPIO connected to the PN532 RSTPDN line.
in	*uartSerial*	HardwareSerial connected to the PN532 TX/RX lines.

Definition at line 37 of file PN532Adapter.cpp.
References _interface, _logger, and _nfc.

5.18.2.5 ~PN532Adapter()

```
PN532Adapter::~PN532Adapter () [override]
```

Release the heap-allocated `Adafruit_PN532` instance.
Definition at line 41 of file PN532Adapter.cpp.
References _nfc.

5.18.2.6 PN532Adapter() [5/5]

```
PN532Adapter::PN532Adapter (
            const PN532Adapter & ) [delete]
```

References PN532Adapter().

5.18.3 Member Function Documentation

5.18.3.1 begin()

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

Initialise the PN532 and probe its firmware version.
Calls `Adafruit_PN532::begin()` then issues `getFirmwareVersion()` as a liveness check.

Returns

`true` if the reader responded to `getFirmwareVersion()`, `false` on a wiring fault or a dead module.

Implements CW_NfcTransport.
Definition at line 48 of file PN532Adapter.cpp.
References _nfc.

5.18.3.2 inListPassiveTarget()

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

Poll for a card in the field.
Single shot — returns immediately after one polling cycle. Callers typically invoke this in a loop with a small `delay()`.

Returns

`true` if a card was selected, `false` otherwise.

Implements CW_NfcTransport.
Definition at line 121 of file PN532Adapter.cpp.
References _nfc.

5.18.3.3 operator=()

```
PN532Adapter & PN532Adapter::operator= (
            const PN532Adapter & ) [delete]
```

References PN532Adapter().

5.18.3.4 printFirmwareVersion()

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

Pretty-print the PN532 firmware version + supported features.
Decodes the raw firmware-version word into IC type (`0x32` = PN532), major/minor revision, and a feature flag set (MIFARE / ISO-DEP / FeliCa) and emits a tree-style summary to the logger.

Returns

`true` if the PN532 responded, `false` otherwise.

Implements CW_NfcTransport.
Definition at line 129 of file PN532Adapter.cpp.
References _logger, _nfc, F, and HEX.

5.18.3.5 resetReader()

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

Reset the PN532 into a clean idle state.
Issues `SAMConfig()` which both clears any pending card session and arms the Secure Access Module. Called by CryptnoxWallet between retries so a stuck reader can recover without a hard reset.
Implements CW_NfcTransport.
Definition at line 125 of file PN532Adapter.cpp.
References _nfc.

5.18.3.6 sendAPDU()

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

Exchange one ISO-DEP APDU with the currently selected card.
Wraps `Adafruit_PN532::inDataExchange()` and adapts its semantics to the CW_NfcTransport contract: `responseLen` is in/out (capacity in, actual length out). The `uint8_t apduLen` already enforces the PN532's 255-byte ISO-DEP frame limit at the type level (HIGH-05).
When `CW_DEBUG_LOGGING` is set, dumps the response as a hex grid to the logger.

Parameters

in	*apdu*	APDU bytes (≤ 255).
in	*apduLen*	Length of apdu in bytes.
out	*response*	Response buffer (caller-allocated).
in,out	*responseLen*	In: capacity of response. Out: number of bytes actually written.

Returns

true on a successful exchange, false on a PN532 / card communication failure.

Implements CW_NfcTransport.
Definition at line 53 of file PN532Adapter.cpp.
References _logger, _nfc, F, and HEX.

5.18.3.7 sendAPDULarge()

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

APDU exchange that can return more than 255 bytes.
Uses Adafruit_PN532's inDataExchange16() (added by the Adafruit_PN532_extended_frame.patch) which parses both PN532 normal and extended frame formats — required by GET_MANUFACTURER_CERTIFICATE pages that exceed the 255-byte normal-frame ceiling on cards whose cert is ∼411 bytes.

Parameters

in	*apdu*	APDU bytes (≤ 255).
in	*apduLen*	Length of apdu in bytes.
out	*response*	Response buffer (caller-allocated).
in,out	*responseLen*	In: capacity of response (up to ∼436). Out: number of bytes actually written.

Returns

true on a successful exchange, false on a PN532 / card communication failure or frame parse error.

Reimplemented from CW_NfcTransport.
Definition at line 87 of file PN532Adapter.cpp.
References _logger, _nfc, F, and HEX.

5.18.4 Member Data Documentation

5.18.4.1 _interface

```
PN532Interface PN532Adapter::_interface  [private]
```

Wiring variant the active constructor selected.
Definition at line 237 of file PN532Adapter.h.
Referenced by PN532Adapter(), PN532Adapter(), PN532Adapter(), and PN532Adapter().

5.18.4.2 _logger

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

Logger reference (non-owning).
Definition at line 236 of file PN532Adapter.h.
Referenced by PN532Adapter(), PN532Adapter(), PN532Adapter(), PN532Adapter(), printFirmwareVersion(), sendAPDU(), and sendAPDULarge().

5.18.4.3 _nfc

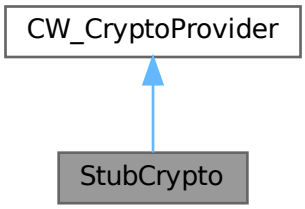
```
Adafruit_PN532* PN532Adapter::_nfc  [private]
```

Owned Adafruit_PN532 driver instance.
Definition at line 238 of file PN532Adapter.h.
Referenced by begin(), inListPassiveTarget(), PN532Adapter(), PN532Adapter(), PN532Adapter(), PN532Adapter(), printFirmwareVersion(), resetReader(), sendAPDU(), sendAPDULarge(), and ~PN532Adapter().
The documentation for this class was generated from the following files:

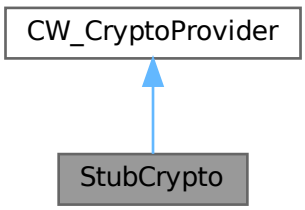
- src/PN532Adapter.h
- src/PN532Adapter.cpp

5.19 StubCrypto Class Reference

Inheritance diagram for StubCrypto:



Collaboration diagram for StubCrypto:



Public Member Functions

- bool sha256 (const uint8_t *, size_t, uint8_t *) override

  Compute SHA-256 over a contiguous data buffer.

- bool sha512 (const uint8_t *, size_t, uint8_t *) override
  Compute SHA-512 over a contiguous data buffer.
- uint16_t aesCbcEncrypt (const uint8_t *, uint16_t, uint8_t *, const uint8_t *, uint8_t, uint8_t *, bool) override
  AES-CBC encrypt.
- uint16_t aesCbcDecrypt (uint8_t *, uint16_t, uint8_t *, const uint8_t *, uint8_t, uint8_t *, bool) override
  AES-CBC decrypt.
- bool ecdh (const uint8_t *, const uint8_t *, uint8_t *, CW_Curve) override
  ECDH shared secret computation.
- bool makeKey (uint8_t *, uint8_t *, CW_Curve) override
  Generate a new EC key pair.
- bool random (uint8_t *, unsigned) override
  Fill a buffer with cryptographically random bytes.
- bool ecdsaVerify (const uint8_t *, const uint8_t *, size_t, const uint8_t *, CW_Curve) override
  Verify an ECDSA signature (raw r||s, 64 bytes) against a message hash.

Public Member Functions inherited from CW_CryptoProvider

- virtual ~CW_CryptoProvider ()

5.19.1 Detailed Description

Definition at line 82 of file fuzz_der.cpp.

5.19.2 Member Function Documentation

5.19.2.1 aesCbcDecrypt()

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

AES-CBC decrypt.

Parameters

in	*in*	Ciphertext input buffer (non-const; may be modified internally).
in	*len*	Length of the ciphertext.
out	*out*	Plaintext output buffer.
in	*key*	AES key bytes.
in	*keyLen*	AES key length in bytes.
in, out	*iv*	16-byte IV used as decrypt IV.
in	*bitPadding*	true = Bit padding removal; false = Null padding (no removal).

Returns

Length of the plaintext written to out.

Implements CW_CryptoProvider.
Definition at line 89 of file fuzz_der.cpp.

5.19.2.2 aesCbcEncrypt()

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

AES-CBC encrypt.

Parameters

in	*in*	Plaintext input buffer.
in	*len*	Length of the plaintext.
out	*out*	Ciphertext output buffer (must be large enough for padding).
in	*key*	AES key bytes.
in	*keyLen*	AES key length in bytes (16, 24, or 32).
in,out	*iv*	16-byte IV; updated to last cipher block on return.
in	*bitPadding*	true = ISO/IEC 9797-1 Method 2 (Bit) padding; false = Null padding (no padding added).

Returns

Length of the ciphertext written to `out`.

Implements CW_CryptoProvider.
Definition at line 86 of file fuzz_der.cpp.

5.19.2.3 ecdh()

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

ECDH shared secret computation.

Parameters

in	*pubKey*	Remote public key (64 bytes, X||Y, no 0x04 prefix).
in	*privKey*	Local private key (32 bytes).
out	*secret*	32-byte shared secret output.
in	*curve*	Curve identifier (CW_CURVE_SECP256R1 or CW_CURVE_SECP256K1).

Returns

true on success, false otherwise.

Implements CW_CryptoProvider.
Definition at line 92 of file fuzz_der.cpp.

5.19.2.4 ecdsaVerify()

```
bool StubCrypto::ecdsaVerify (
```

```
            const uint8_t * pubKey64,
            const uint8_t * hash,
            size_t hashLen,
            const uint8_t * sig,
            CW_Curve curve)  [inline], [override], [virtual]
```

Verify an ECDSA signature (raw r||s, 64 bytes) against a message hash.

Parameters

in	*pubKey64*	64-byte public key (X||Y, no 0x04 prefix).
in	*hash*	Message hash buffer.
in	*hashLen*	Length of the hash in bytes.
in	*sig*	64-byte raw signature (r[32]||s[32]).
in	*curve*	Curve identifier for the verification operation.

Returns

true if the signature is valid, false otherwise.

Implements CW_CryptoProvider.
Definition at line 97 of file fuzz_der.cpp.

5.19.2.5 makeKey()

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

Generate a new EC key pair.

Parameters

out	*pubKey*	64-byte public key output (X||Y, no prefix).
out	*privKey*	32-byte private key output.
in	*curve*	Curve identifier (CW_CURVE_SECP256R1 or CW_CURVE_SECP256K1).

Returns

true on success, false otherwise.

Implements CW_CryptoProvider.
Definition at line 94 of file fuzz_der.cpp.

5.19.2.6 random()

```
bool StubCrypto::random (
            uint8_t * dest,
            unsigned size)  [inline], [override], [virtual]
```

Fill a buffer with cryptographically random bytes.

Parameters

out	*dest*	Buffer to fill.
in	*size*	Number of bytes to generate.

Returns

true on success, false otherwise.

Implements CW_CryptoProvider.
Definition at line 96 of file fuzz_der.cpp.

5.19.2.7 sha256()

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

Compute SHA-256 over a contiguous data buffer.

Parameters

in	*data*	Input buffer.
in	*len*	Number of bytes to hash.
out	*out*	32-byte output buffer.

Returns

true on success, false if the underlying hash accelerator faults.

Implements CW_CryptoProvider.
Definition at line 84 of file fuzz_der.cpp.

5.19.2.8 sha512()

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

Compute SHA-512 over a contiguous data buffer.

Parameters

in	*data*	Input buffer.
in	*len*	Number of bytes to hash.
out	*out*	64-byte output buffer.

Returns

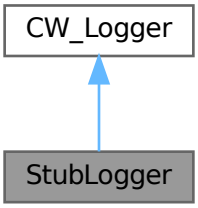
true on success, false if the underlying hash accelerator faults.

Implements CW_CryptoProvider.
Definition at line 85 of file fuzz_der.cpp.
The documentation for this class was generated from the following file:

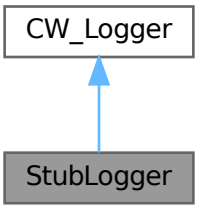
- src/cryptnox-sdk-cpp/fuzz/fuzz_der.cpp

5.20 StubLogger Class Reference

Inheritance diagram for StubLogger:



Collaboration diagram for StubLogger:



Public Member Functions

- bool begin (unsigned long) override
  Initialize the logging interface.
- void print (const __FlashStringHelper *) override
- void print (const char *) override
- void print (char) override
- void print (uint8_t, int) override
- void print (uint16_t, int) override
- void print (uint32_t, int) override
- void print (int, int) override
- void println () override
- void println (const __FlashStringHelper *) override
- void println (const char *) override
- void println (char) override
- void println (uint8_t, int) override
- void println (uint16_t, int) override
- void println (uint32_t, int) override
- void println (int, int) override

Public Member Functions inherited from CW_Logger

- virtual ~CW_Logger ()

5.20.1 Detailed Description

Definition at line 62 of file fuzz_der.cpp.

5.20.2 Member Function Documentation

5.20.2.1 begin()

```
bool StubLogger::begin (
            unsigned long baudRate)  [inline], [override], [virtual]
```

Initialize the logging interface.

Parameters

baudRate	Baud rate (relevant for UART implementations).

Returns

true if initialization succeeded, false otherwise.

Implements CW_Logger.
Definition at line 64 of file fuzz_der.cpp.

5.20.2.2 print() [1/7]

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

Implements CW_Logger.
Definition at line 67 of file fuzz_der.cpp.

5.20.2.3 print() [2/7]

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

Implements CW_Logger.
Definition at line 65 of file fuzz_der.cpp.

5.20.2.4 print() [3/7]

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

Implements CW_Logger.
Definition at line 66 of file fuzz_der.cpp.

5.20.2.5 print() [4/7]

```
void StubLogger::print (
            int ,
            int )  [inline], [override], [virtual]
```

Implements CW_Logger.
Definition at line 71 of file fuzz_der.cpp.

5.20.2.6 print() [5/7]

```
void StubLogger::print (
            uint16_t ,
            int )  [inline], [override], [virtual]
```

Implements CW_Logger.
Definition at line 69 of file fuzz_der.cpp.

5.20.2.7 print() [6/7]

```
void StubLogger::print (
            uint32_t ,
            int )  [inline], [override], [virtual]
```

Implements CW_Logger.
Definition at line 70 of file fuzz_der.cpp.

5.20.2.8 print() [7/7]

```
void StubLogger::print (
            uint8_t ,
            int )  [inline], [override], [virtual]
```

Implements CW_Logger.
Definition at line 68 of file fuzz_der.cpp.

5.20.2.9 println() [1/8]

```
void StubLogger::println ()  [inline], [override], [virtual]
```

Implements CW_Logger.
Definition at line 72 of file fuzz_der.cpp.

5.20.2.10 println() [2/8]

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

Implements CW_Logger.
Definition at line 75 of file fuzz_der.cpp.

5.20.2.11 println() [3/8]

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

Implements CW_Logger.
Definition at line 73 of file fuzz_der.cpp.

5.20.2.12 println() [4/8]

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

Implements CW_Logger.
Definition at line 74 of file fuzz_der.cpp.

5.20.2.13 println() [5/8]

```
void StubLogger::println (
            int ,
            int )  [inline], [override], [virtual]
```

Implements CW_Logger.
Definition at line 79 of file fuzz_der.cpp.

5.20.2.14 println() [6/8]

```
void StubLogger::println (
            uint16_t ,
            int )  [inline], [override], [virtual]
```

Implements CW_Logger.
Definition at line 77 of file fuzz_der.cpp.

5.20.2.15 println() [7/8]

```
void StubLogger::println (
            uint32_t ,
            int )  [inline], [override], [virtual]
```

Implements CW_Logger.
Definition at line 78 of file fuzz_der.cpp.

5.20.2.16 println() [8/8]

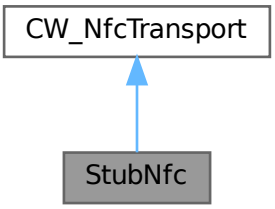
```
void StubLogger::println (
            uint8_t ,
            int )  [inline], [override], [virtual]
```

Implements CW_Logger.
Definition at line 76 of file fuzz_der.cpp.
The documentation for this class was generated from the following file:

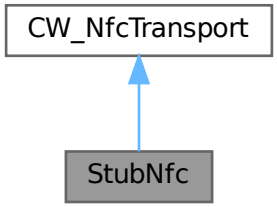
- src/cryptnox-sdk-cpp/fuzz/fuzz_der.cpp

5.21 StubNfc Class Reference

Inheritance diagram for StubNfc:



Collaboration diagram for StubNfc:



Public Member Functions

- bool begin () override

  Initialize the NFC transport hardware.
- bool inListPassiveTarget () override

  Detect the presence of a passive ISO-DEP NFC target.
- bool sendAPDU (const uint8_t *, uint8_t, uint8_t *, uint8_t &) override

  Send an APDU command to the card and receive the response.
- void resetReader () override

  Reset the NFC reader/field for the next card detection cycle.
- bool printFirmwareVersion () override

  Print NFC module firmware version information to the logger.

Public Member Functions inherited from CW_NfcTransport

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

  Send an APDU and receive a response that may exceed 255 bytes.
- virtual ~CW_NfcTransport ()

5.21.1 Detailed Description

Definition at line 52 of file fuzz_der.cpp.

5.21.2 Member Function Documentation

5.21.2.1 begin()

```
bool StubNfc::begin ()  [inline], [override], [virtual]
```

Initialize the NFC transport hardware.

Returns

true if initialization succeeded, false otherwise.

Implements CW_NfcTransport.
Definition at line 54 of file fuzz_der.cpp.

5.21.2.2 inListPassiveTarget()

```
bool StubNfc::inListPassiveTarget () [inline], [override], [virtual]
```

Detect the presence of a passive ISO-DEP NFC target.

Returns

true if a card is detected, false otherwise.

Implements CW_NfcTransport.
Definition at line 55 of file fuzz_der.cpp.

5.21.2.3 printFirmwareVersion()

```
bool StubNfc::printFirmwareVersion () [inline], [override], [virtual]
```

Print NFC module firmware version information to the logger.

Returns

true if firmware info was retrieved successfully, false otherwise.

Implements CW_NfcTransport.
Definition at line 59 of file fuzz_der.cpp.

5.21.2.4 resetReader()

```
void StubNfc::resetReader () [inline], [override], [virtual]
```

Reset the NFC reader/field for the next card detection cycle.
Implements CW_NfcTransport.
Definition at line 58 of file fuzz_der.cpp.

5.21.2.5 sendAPDU()

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

Send an APDU command to the card and receive the response.

Parameters

in	*apdu*	APDU command bytes.
in	*apduLen*	Length of the APDU command.
out	*response*	Buffer to receive the card response.
out	*responseLen*	Actual number of bytes written to `response`.

Returns

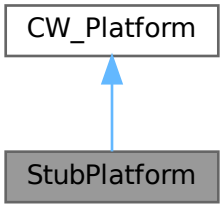
true if the exchange succeeded, false otherwise.

Implements CW_NfcTransport.
Definition at line 56 of file fuzz_der.cpp.
The documentation for this class was generated from the following file:

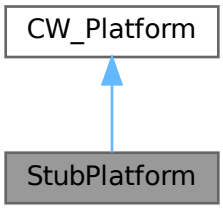
- src/cryptnox-sdk-cpp/fuzz/fuzz_der.cpp

5.22 StubPlatform Class Reference

Inheritance diagram for StubPlatform:



Collaboration diagram for StubPlatform:



Public Member Functions

- void sleep_ms (uint32_t) override
  Block for at least `ms` *milliseconds.*

Public Member Functions inherited from CW_Platform

- virtual ~CW_Platform ()

5.22.1 Detailed Description

Definition at line 101 of file fuzz_der.cpp.

5.22.2 Member Function Documentation

5.22.2.1 sleep_ms()

```
void StubPlatform::sleep_ms (
            uint32_t ms)  [inline], [override], [virtual]
```

Block for at least `ms` milliseconds.

Parameters

in	*ms*	Duration to sleep in milliseconds.

Implements CW_Platform.
Definition at line 103 of file fuzz_der.cpp.
The documentation for this class was generated from the following file:

- src/cryptnox-sdk-cpp/fuzz/fuzz_der.cpp

5.23 Tx2 Struct Reference

Ethereum EIP-1559 transaction structure.

Public Attributes

- uint64_t nonce
- uint64_t maxPriorityFeePerGas
- uint64_t maxFeePerGas
- uint64_t gasLimit
- const char * to
- uint64_t value
- const uint8_t * data
- size_t dataLen
- uint32_t chainId

5.23.1 Detailed Description

Ethereum EIP-1559 transaction structure.

Examples

UsdcSigning.ino.

Definition at line 133 of file UsdcSigning.ino.

5.23.2 Member Data Documentation

5.23.2.1 chainId

`uint32_t Tx2::chainId`
Ethereum chain ID

Examples

UsdcSigning.ino.

Definition at line 142 of file UsdcSigning.ino.
Referenced by rlpEncodeTxBody(), and setup().

5.23.2.2 data

`const uint8_t* Tx2::data`
Transaction calldata

Examples

UsdcSigning.ino.

Definition at line 140 of file UsdcSigning.ino.
Referenced by rlpEncodeTxBody(), and setup().

5.23.2.3 dataLen

`size_t Tx2::dataLen`
Length of calldata

Examples

UsdcSigning.ino.

Definition at line 141 of file UsdcSigning.ino.
Referenced by rlpEncodeTxBody(), and setup().

5.23.2.4 gasLimit

`uint64_t Tx2::gasLimit`
Gas limit

Examples

UsdcSigning.ino.

Definition at line 137 of file UsdcSigning.ino.
Referenced by rlpEncodeTxBody(), and setup().

5.23.2.5 maxFeePerGas

`uint64_t Tx2::maxFeePerGas`
Max fee (wei)

Examples

UsdcSigning.ino.

Definition at line 136 of file UsdcSigning.ino.
Referenced by rlpEncodeTxBody(), and setup().

5.23.2.6 maxPriorityFeePerGas

`uint64_t Tx2::maxPriorityFeePerGas`
Max priority fee (wei)

Examples

UsdcSigning.ino.

Definition at line 135 of file UsdcSigning.ino.
Referenced by rlpEncodeTxBody(), and setup().

5.23.2.7 nonce

`uint64_t Tx2::nonce`
Transaction nonce

Examples

UsdcSigning.ino.

Definition at line 134 of file UsdcSigning.ino.
Referenced by rlpEncodeTxBody(), and setup().

5.23.2.8 to

`const char* Tx2::to`
Recipient address

Examples

UsdcSigning.ino.

Definition at line 138 of file UsdcSigning.ino.
Referenced by rlpEncodeTxBody(), and setup().

5.23.2.9 value

`uint64_t Tx2::value`
Value in wei

Examples

UsdcSigning.ino.

Definition at line 139 of file UsdcSigning.ino.
Referenced by rlpEncodeTxBody(), and setup().
The documentation for this struct was generated from the following file:

- examples/UsdcSigning/UsdcSigning.ino

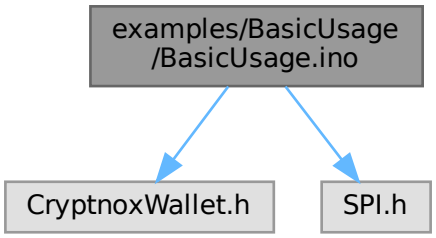
Chapter 6

File Documentation

6.1 examples/BasicUsage/BasicUsage.ino File Reference

```
#include <CryptnoxWallet.h>
#include <SPI.h>
```

Include dependency graph for BasicUsage.ino:



Macros

- #define USE_SPI
- #define PN532_SS (10U)

  SPI slave select (CS) pin for the PN532 module.
- #define DEFAULT_PIN "000000000"

  Default PIN code (ASCII digits). Must match the PIN used during card.init().
- #define DEFAULT_PIN_LEN (sizeof(DEFAULT_PIN) - 1U)

Functions

- PN532Adapter nfc (serialAdapter, PN532_SS, &SPI)
- void setup ()

  Arduino setup function.
- void loop ()

  Arduino main loop.

Variables

- ArduinoLoggerAdapter serialAdapter

- ArduinoCryptoProvider cryptoProvider
- ArduinoPlatform platform
- CryptnoxWallet wallet (nfc, serialAdapter, cryptoProvider, platform)

6.1.1 Macro Definition Documentation

6.1.1.1 DEFAULT_PIN

```
#define DEFAULT_PIN "000000000"
```

Default PIN code (ASCII digits). Must match the PIN used during card.init().

Examples

BasicUsage.ino.

Definition at line 75 of file BasicUsage.ino.
Referenced by loop().

6.1.1.2 DEFAULT_PIN_LEN

```
#define DEFAULT_PIN_LEN (sizeof(DEFAULT_PIN) - 1U)
```

Examples

BasicUsage.ino.

Definition at line 76 of file BasicUsage.ino.
Referenced by loop().

6.1.1.3 PN532_SS

```
#define PN532_SS (10U)
```

SPI slave select (CS) pin for the PN532 module.

Examples

BasicUsage.ino.

Definition at line 39 of file BasicUsage.ino.
Referenced by nfc().

6.1.1.4 USE_SPI

```
#define USE_SPI
```

Definition at line 25 of file BasicUsage.ino.

6.1.2 Function Documentation

6.1.2.1 loop()

```
void loop ()
```

Arduino main loop.
Demonstrates simplified card connection and processing:

1. Connect to card and establish secure channel (combines detection and channel setup)
2. Sign a test hash (PIN included in sign payload)
3. Clear session and reset reader

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 117 of file BasicUsage.ino.
References CW_HASH_SIZE, CW_OK, CW_SIG_R_OFFSET, CW_SIG_S_OFFSET, CW_SIGN_CURR_K1, CW_SIGN_SIG_ECDSA_LOW_S, CW_SIGN_WITH_PIN, DEFAULT_PIN, DEFAULT_PIN_LEN, CW_SignResult::errorCode, F, CW_SignRequest::hash, CW_SignRequest::hashLength, HEX, CW_SignRequest::pin, CW_Utils::safe_memcpy(), CW_Utils::secure_wipe(), serialAdapter, CW_SignResult::signature, and wallet.

6.1.2.2 nfc()

```
PN532Adapter nfc (
            serialAdapter ,
            PN532_SS ,
            & SPI)
```

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

References PN532_SS, and serialAdapter.
Referenced by setup().

6.1.2.3 setup()

```
void setup ()
```

Arduino setup function.
Initializes the serial port for debugging, the selected bus, and the PN532 module.

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 88 of file BasicUsage.ino.
References F, serialAdapter, and wallet.

6.1.3 Variable Documentation

6.1.3.1 cryptoProvider

```
ArduinoCryptoProvider cryptoProvider
```

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 78 of file BasicUsage.ino.

6.1.3.2 platform

```
ArduinoPlatform platform
```

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 79 of file BasicUsage.ino.
Referenced by CryptnoxWallet::CryptnoxWallet(), and CW_SecureChannel::CW_SecureChannel().

6.1.3.3 serialAdapter

```
ArduinoLoggerAdapter serialAdapter
```

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 47 of file BasicUsage.ino.
Referenced by loop(), nfc(), nfc(), and setup().

6.1.3.4 wallet

```
CryptnoxWallet wallet(nfc, serialAdapter, cryptoProvider, platform) (
             nfc ,
             serialAdapter ,
             cryptoProvider ,
             platform )
```

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Referenced by loop(), and setup().

6.2 BasicUsage.ino

Go to the documentation of this file.

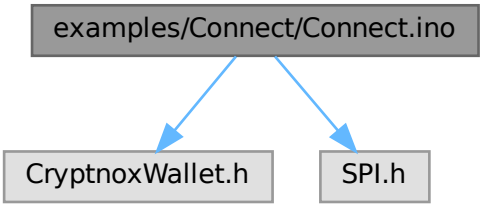
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00017
00018 #include <CryptnoxWallet.h>
00019
00020 /* ========================================================================
00021  * 1. Interface Selection
00022  * ------------------------------------------------------------------------
00023  * Uncomment exactly ONE of the two lines below to choose the PN532 interface.
00024  * ======================================================================== */
00025 #define USE_SPI
00026 // #define USE_I2C
00027
00028 /* ========================================================================
00029  * 2. Pin Configuration
00030  * ======================================================================== */
00031 #if defined(USE_SPI)
00032
00033 #include <SPI.h>
00034
00039 #define PN532_SS    (10U)
00040
00041 /* PRODUCTION NOTE (LOW-03): ArduinoLoggerAdapter streams all log output over
00042  * USB-CDC Serial, which is visible to any attached host process.  For firmware
00043  * that is shipped to end-users, replace the two lines below with:
00044  *   NullLoggerAdapter serialAdapter;
00045  * and remove the serialAdapter.begin() call in setup().  NullLoggerAdapter is
00046  * a no-op and incurs zero code-size or runtime overhead. */
00047 ArduinoLoggerAdapter serialAdapter;
00048 PN532Adapter nfc(serialAdapter, PN532_SS, &SPI);
00049
00050 #elif defined(USE_I2C)
00051
00052 #include <Wire.h>
00053
00058 #define PN532_IRQ   (2U)
00059
00064 #define PN532_RST   (3U)
00065
00066 /* PRODUCTION NOTE (LOW-03): see comment in the USE_SPI block above. */
00067 ArduinoLoggerAdapter serialAdapter;
00068 PN532Adapter nfc(serialAdapter, PN532_IRQ, PN532_RST, &Wire);
00069
00070 #else
00071 #error "Please define USE_SPI or USE_I2C to select the PN532 interface."
00072 #endif
00073
00075 #define DEFAULT_PIN       "000000000"
00076 #define DEFAULT_PIN_LEN   (sizeof(DEFAULT_PIN) - 1U)
00077
00078 ArduinoCryptoProvider cryptoProvider;
00079 ArduinoPlatform       platform;
00080 CryptnoxWallet wallet(nfc, serialAdapter, cryptoProvider, platform);
00081
00088 void setup() {
00089     serialAdapter.begin(115200);
00090
00091     /* Arduino R4: Wait 1s to get Serial ready */
00092     delay(1000);
00093
00094 #if defined(USE_SPI)
```

```
00095     /* Initialize SPI bus */
00096     SPI.begin();
00097 #elif defined(USE_I2C)
00098     /* Initialize I2C bus */
00099     Wire.begin();
00100 #endif
00101
00102     /* Initialize the PN532 module -- halt on failure to avoid silent errors */
00103     if (!wallet.begin()) {
00104         serialAdapter.println(F("PN532 init failed"));
00105         while(1);
00106     }
00107 }
00108
00117 void loop() {
00118
00119     /* Step 1: Connect to card and establish secure channel */
00120     CW_SecureSession session;
00121     if (wallet.connect(session)) {
00122         serialAdapter.println(F("Card connected and secure channel established"));
00123
00124         /* Step 2: Sign a test hash (32 bytes of 0x01 for demo purposes) */
00125         /* NOTE: Card must have a seed loaded (via Python SDK: card.generate_seed(pin) */
00126         /*       or card.load_seed(seed, pin)) before signing will work.              */
00127         serialAdapter.println(F("Signing test hash..."));
00128         uint8_t testHash[CW_HASH_SIZE];
00129         memset(testHash, 0x01, sizeof(testHash));
00130
00131         /* Build sign request per CW_SignRequest API.
00132          * PIN is included in the sign data payload for authentication.
00133          * Alternatively, call verifyPin() first and omit the PIN here. */
00134         CW_SignRequest signRequest(session, CW_SIGN_CURR_K1, CW_SIGN_SIG_ECDSA_LOW_S,
      CW_SIGN_WITH_PIN);
00135         signRequest.hash = testHash;
00136         signRequest.hashLength = sizeof(testHash);
00137         /* Set PIN (must match the PIN used during card.init()) */
00138         CW_Utils::safe_memcpy(signRequest.pin, sizeof(signRequest.pin),
00139                               reinterpret_cast<const uint8_t*>(DEFAULT_PIN), DEFAULT_PIN_LEN);
00140
00141         CW_SignResult signResult = wallet.sign(signRequest);
00142
00143         if (signResult.errorCode == CW_OK) {
00144             serialAdapter.println(F("Signature received (64 bytes raw r||s)"));
00145
00146             /* Print first 8 bytes of R and S for quick visual check */
00147             serialAdapter.print(F("  R[0..7]: "));
00148             for (uint8_t i = CW_SIG_R_OFFSET; i < CW_SIG_R_OFFSET + 8U; i++) {
00149                 if (signResult.signature[i] < 0x10U) serialAdapter.print(F("0"));
00150                 serialAdapter.print(signResult.signature[i], HEX);
00151                 serialAdapter.print(F(" "));
00152             }
00153             serialAdapter.println();
00154             serialAdapter.print(F("  S[0..7]: "));
00155             for (uint8_t i = CW_SIG_S_OFFSET; i < CW_SIG_S_OFFSET + 8U; i++) {
00156                 if (signResult.signature[i] < 0x10U) serialAdapter.print(F("0"));
00157                 serialAdapter.print(signResult.signature[i], HEX);
00158                 serialAdapter.print(F(" "));
00159             }
00160             serialAdapter.println();
00161             serialAdapter.println(F("Card processed successfully"));
00162         } else {
00163             serialAdapter.print(F("Sign failed, errorCode: 0x"));
00164             serialAdapter.println(signResult.errorCode, HEX);
00165         }
00166
00167         /* Securely wipe sensitive buffers */
00168         CW_Utils::secure_wipe(testHash, sizeof(testHash));
00169         CW_Utils::secure_wipe(signResult.signature, sizeof(signResult.signature));
00170     }
00171
00172     /* Always disconnect to reset reader for next card detection */
00173     wallet.disconnect(session);
00174
00175     /* Wait before next iteration */
00176     delay(1000);
00177 }
```

6.3 examples/Connect/Connect.ino File Reference

```
#include <CryptnoxWallet.h>
#include <SPI.h>
```

Include dependency graph for Connect.ino:



Macros

- #define PN532_SS_PIN (10U)
  SPI slave-select (CS) pin connected to the PN532 module.

Functions

- PN532Adapter nfc (serialAdapter, PN532_SS_PIN, &SPI)
  PN532 transport adapter over SPI.
- void setup ()
  Arduino setup hook.
- void loop ()
  Arduino main loop.

Variables

- ArduinoLoggerAdapter serialAdapter
  Arduino logger adapter — emits SDK diagnostics on `Serial`.
- ArduinoCryptoProvider cryptoProvider
  Crypto provider (AES / SHA / micro-ecc / TRNG bridge for Arduino).
- ArduinoPlatform platform
  Platform adapter (Arduino blocking delay).
- CryptnoxWallet wallet (nfc, serialAdapter, cryptoProvider, platform)
  High-level Cryptnox wallet wiring the four adapters together.

6.3.1 Macro Definition Documentation

6.3.1.1 PN532_SS_PIN

```
#define PN532_SS_PIN (10U)
```

SPI slave-select (CS) pin connected to the PN532 module.

Examples

Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 31 of file Connect.ino.
Referenced by nfc().

6.3.2 Function Documentation

6.3.2.1 loop()

```
void loop ()
```

Arduino main loop.
One full secure-channel session per iteration:

- CryptnoxWallet::connect performs SELECT + manufacturer cert verification + ECDH key agreement + mutual authentication. When it returns `true` the channel is mathematically established (`session.aesKey`, `session.macKey`, `session.iv` are populated and authenticated against the card).
- CryptnoxWallet::getCardInfo fetches card identity over the channel; on enabled debug logging it prints the card info bytes.
- CryptnoxWallet::disconnect tears the channel down and zeroes the session keys.

The 1 s delay between iterations gives time to remove/replace the card and keeps the Serial output readable.
Definition at line 88 of file Connect.ino.
References CW_CardInfo::email, F, CW_CardInfo::name, serialAdapter, and wallet.

6.3.2.2 nfc()

```
PN532Adapter nfc (
            serialAdapter ,
            PN532_SS_PIN ,
            & SPI)
```

PN532 transport adapter over SPI.
References PN532_SS_PIN, and serialAdapter.

6.3.2.3 setup()

```
void setup ()
```

Arduino setup hook.
Brings up `Serial`, the SPI bus and the PN532 reader, then prints the PN532 firmware version as a sanity check that the reader is alive and speaking. Halts on init failure so the user can inspect the Serial output.
Definition at line 56 of file Connect.ino.
References F, nfc(), serialAdapter, and wallet.

6.3.3 Variable Documentation

6.3.3.1 cryptoProvider

```
ArduinoCryptoProvider cryptoProvider
```

Crypto provider (AES / SHA / micro-ecc / TRNG bridge for Arduino).
Definition at line 40 of file Connect.ino.

6.3.3.2 platform

```
ArduinoPlatform platform
```

Platform adapter (Arduino blocking delay).
Definition at line 43 of file Connect.ino.

6.3.3.3 serialAdapter

```
ArduinoLoggerAdapter serialAdapter
```

Arduino logger adapter — emits SDK diagnostics on `Serial`.
Definition at line 34 of file Connect.ino.

6.3.3.4 wallet

```
CryptnoxWallet wallet(nfc, serialAdapter, cryptoProvider, platform) (
             nfc ,
             serialAdapter ,
             cryptoProvider ,
             platform )
```

High-level Cryptnox wallet wiring the four adapters together.

6.4 Connect.ino

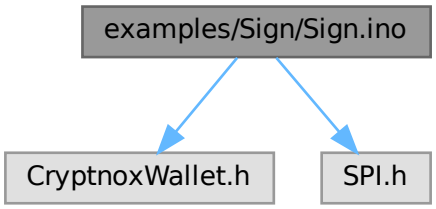
Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00027 #include <CryptnoxWallet.h>
00028 #include <SPI.h>
00029
00031 #define PN532_SS_PIN  (10U)
00032
00034 ArduinoLoggerAdapter   serialAdapter;
00035
00037 PN532Adapter           nfc(serialAdapter, PN532_SS_PIN, &SPI);
00038
00040 ArduinoCryptoProvider  cryptoProvider;
00041
00043 ArduinoPlatform        platform;
00044
00046 CryptnoxWallet         wallet(nfc, serialAdapter, cryptoProvider, platform);
00047
00056 void setup() {
00057     serialAdapter.begin(115200);
00058     delay(1000);                              /* Arduino R4: wait for Serial */
00059     SPI.begin();
00060
00061     if (!wallet.begin()) {
00062         serialAdapter.println(F("PN532 init failed"));
00063         while (1);
00064     }
00065
00066     /* Concrete proof the PN532 is up: prints IC chip, firmware version
00067      * and supported NFC features (MIFARE / ISO-DEP / FeliCa). */
00068     nfc.printFirmwareVersion();
00069 }
00070
00088 void loop() {
00089     CW_SecureSession session;
00090
00091     if (wallet.connect(session)) {
00092         serialAdapter.println(F("Card connected, secure channel established"));
00093
00094         CW_CardInfo info;
00095         if (wallet.getCardInfo(session, &info)) {
00096             serialAdapter.print(F("Owner name : "));
00097             serialAdapter.println(info.name);
00098             serialAdapter.print(F("Owner email: "));
00099             serialAdapter.println(info.email);
00100         } else {
00101             serialAdapter.println(F("getCardInfo failed (channel error or parse error)"));
00102         }
00103     } else {
00104         serialAdapter.println(F("Card not detected or secure channel failed"));
00105     }
00106
00107     wallet.disconnect(session);
00108     delay(1000);
00109 }
```

6.5 examples/Sign/Sign.ino File Reference

```
#include <CryptnoxWallet.h>
#include <SPI.h>
```

Include dependency graph for Sign.ino:



Macros

- #define PN532_SS_PIN (10U)
  SPI slave-select (CS) pin connected to the PN532 module.
- #define DEMO_PIN "000000000"
  Demo PIN used by this example. Must match the PIN that the card was initialised with (4–9 ASCII digits).

Functions

- PN532Adapter nfc (serialAdapter, PN532_SS_PIN, &SPI)
  PN532 transport adapter over SPI.
- void setup ()
  Arduino setup hook.
- void loop ()
  Arduino main loop.

Variables

- ArduinoLoggerAdapter serialAdapter
  Arduino logger adapter — emits SDK diagnostics on `Serial`.
- ArduinoCryptoProvider cryptoProvider
  Crypto provider (AES / SHA / micro-ecc / TRNG bridge for Arduino).
- ArduinoPlatform platform
  Platform adapter (Arduino blocking delay).
- CryptnoxWallet wallet (nfc, serialAdapter, cryptoProvider, platform)
  High-level Cryptnox wallet wiring the four adapters together.

6.5.1 Macro Definition Documentation

6.5.1.1 DEMO_PIN

```
#define DEMO_PIN "000000000"
```

Demo PIN used by this example. Must match the PIN that the card was initialised with (4–9 ASCII digits).

Examples

Sign.ino, and VerifyPin.ino.

Definition at line 50 of file Sign.ino.
Referenced by loop().

6.5.1.2 PN532_SS_PIN

```
#define PN532_SS_PIN (10U)
```
SPI slave-select (CS) pin connected to the PN532 module.
Definition at line 44 of file Sign.ino.

6.5.2 Function Documentation

6.5.2.1 loop()

```
void loop ()
```
Arduino main loop.
One full sign session per iteration on the happy path:

- CryptnoxWallet::connect opens the secure channel,
- CryptnoxWallet::sign signs the test hash with the on-card key,
- sensitive stack buffers are wiped via CW_Utils::secure_wipe,
- CryptnoxWallet::disconnect tears the channel down.

On CW_SIGN_PIN_INCORRECT the sketch enters an infinite halt to protect the card's retry counter. Other non-OK error codes are reported but the loop continues (those errors do not consume PIN tries). The 1 s delay between successful iterations gives time to remove/replace the card and keeps the Serial output readable.
Definition at line 100 of file Sign.ino.
References CW_HASH_SIZE, CW_OK, CW_SIG_R_OFFSET, CW_SIG_S_OFFSET, CW_SIGN_CURR_K1, CW_SIGN_PIN_INCORRECT, CW_SIGN_SIG_ECDSA_LOW_S, CW_SIGN_WITH_PIN, DEMO_PIN, CW_SignResult::errorCode, F, CW_SignRequest::hash, CW_SignRequest::hashLength, HEX, CW_SignRequest::pin, CW_Utils::safe_memcpy(), CW_Utils::secure_wipe(), serialAdapter, CW_SignResult::signature, and wallet.

6.5.2.2 nfc()

```
PN532Adapter nfc (
            serialAdapter ,
            PN532_SS_PIN ,
            & SPI)
```
PN532 transport adapter over SPI.
References PN532_SS_PIN, and serialAdapter.

6.5.2.3 setup()

```
void setup ()
```
Arduino setup hook.
Brings up `Serial`, the SPI bus and the PN532 reader. Halts on init failure (no reader detected) so the user can inspect the Serial output.
Definition at line 73 of file Sign.ino.
References F, serialAdapter, and wallet.

6.5.3 Variable Documentation

6.5.3.1 cryptoProvider

```
ArduinoCryptoProvider cryptoProvider
```
Crypto provider (AES / SHA / micro-ecc / TRNG bridge for Arduino).
Definition at line 59 of file Sign.ino.

6.5.3.2 platform

`ArduinoPlatform platform`
Platform adapter (Arduino blocking delay).
Definition at line 62 of file Sign.ino.

6.5.3.3 serialAdapter

`ArduinoLoggerAdapter serialAdapter`
Arduino logger adapter — emits SDK diagnostics on `Serial`.
Definition at line 53 of file Sign.ino.

6.5.3.4 wallet

```
CryptnoxWallet wallet(nfc, serialAdapter, cryptoProvider, platform) (
             nfc ,
             serialAdapter ,
             cryptoProvider ,
             platform )
```

High-level Cryptnox wallet wiring the four adapters together.

6.6 Sign.ino

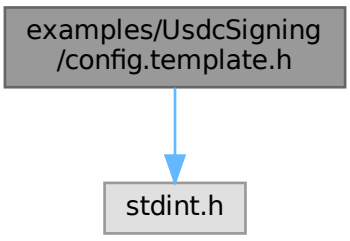
Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00040 #include <CryptnoxWallet.h>
00041 #include <SPI.h>
00042
00044 #define PN532_SS_PIN  (10U)
00045
00050 #define DEMO_PIN      "000000000"
00051
00053 ArduinoLoggerAdapter   serialAdapter;
00054
00056 PN532Adapter           nfc(serialAdapter, PN532_SS_PIN, &SPI);
00057
00059 ArduinoCryptoProvider  cryptoProvider;
00060
00062 ArduinoPlatform        platform;
00063
00065 CryptnoxWallet         wallet(nfc, serialAdapter, cryptoProvider, platform);
00066
00073 void setup() {
00074     serialAdapter.begin(115200);
00075     delay(1000);                              /* Arduino R4: wait for Serial */
00076     SPI.begin();
00077
00078     if (!wallet.begin()) {
00079         serialAdapter.println(F("PN532 init failed"));
00080         while (1);
00081     }
00082 }
00083
00100 void loop() {
00101     CW_SecureSession session;
00102     if (!wallet.connect(session)) {
00103         serialAdapter.println(F("Card not detected"));
00104         wallet.disconnect(session);
00105         delay(1000);
00106         return;
00107     }
00108
00109     uint8_t hash[CW_HASH_SIZE];
00110     memset(hash, 0x01, sizeof(hash));   /* replace with SHA-256 of your tx */
00111
00112     CW_SignRequest req(session,
00113                        CW_SIGN_CURR_K1,
00114                        CW_SIGN_SIG_ECDSA_LOW_S,
00115                        CW_SIGN_WITH_PIN);
00116     req.hash       = hash;
00117     req.hashLength = sizeof(hash);
```

```
00118     CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
00119                           reinterpret_cast<const uint8_t*>(DEMO_PIN), strlen(DEMO_PIN));
00120
00121     CW_SignResult sig = wallet.sign(req);
00122     if (sig.errorCode == CW_OK) {
00123         serialAdapter.println(F("Signature OK"));
00124
00125         serialAdapter.print(F("  r = "));
00126         for (uint8_t i = 0U; i < 32U; i++) {
00127             uint8_t b = sig.signature[CW_SIG_R_OFFSET + i];
00128             if (b < 0x10U) { serialAdapter.print(F("0")); }
00129             serialAdapter.print(b, HEX);
00130         }
00131         serialAdapter.println();
00132
00133         serialAdapter.print(F("  s = "));
00134         for (uint8_t i = 0U; i < 32U; i++) {
00135             uint8_t b = sig.signature[CW_SIG_S_OFFSET + i];
00136             if (b < 0x10U) { serialAdapter.print(F("0")); }
00137             serialAdapter.print(b, HEX);
00138         }
00139         serialAdapter.println();
00140
00141         serialAdapter.print(F("  errorCode = 0x"));
00142         serialAdapter.println(sig.errorCode, HEX);
00143     } else if (sig.errorCode == CW_SIGN_PIN_INCORRECT) {
00144         /* CRITICAL: do NOT loop -- each wrong PIN attempt burns a retry. */
00145         serialAdapter.println(F("Wrong PIN -- halting to protect retry counter"));
00146         CW_Utils::secure_wipe(hash, sizeof(hash));
00147         CW_Utils::secure_wipe(sig.signature, sizeof(sig.signature));
00148         wallet.disconnect(session);
00149         while (1);
00150     } else {
00151         serialAdapter.print(F("Sign failed: 0x"));
00152         serialAdapter.println(sig.errorCode, HEX);
00153     }
00154
00155     CW_Utils::secure_wipe(hash, sizeof(hash));
00156     CW_Utils::secure_wipe(sig.signature, sizeof(sig.signature));
00157     wallet.disconnect(session);
00158     delay(1000);
00159 }
```

6.7 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_PATH "/"

- #define WIFI_CA_CERT
- #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

6.7.1 Macro Definition Documentation

6.7.1.1 ADDR_FROM

```
#define ADDR_FROM "<SENDER_ADDRESS>"
```

Examples

UsdcSigning.ino.

Definition at line 101 of file config.template.h.
Referenced by determineYParity(), and fetchNonce().

6.7.1.2 ADDR_TO

```
#define ADDR_TO "<RECIPIENT_ADDRESS>"
```

Examples

UsdcSigning.ino.

Definition at line 104 of file config.template.h.
Referenced by encodeERC20Transfer().

6.7.1.3 ADDR_USDC

```
#define ADDR_USDC "<USDC_CONTRACT_ADDRESS>"
```

Examples

UsdcSigning.ino.

Definition at line 107 of file config.template.h.
Referenced by setup().

6.7.1.4 AMOUNT_USDC

```
#define AMOUNT_USDC 1000000UL /* 1.0 USDC */
```

Examples

UsdcSigning.ino.

Definition at line 115 of file config.template.h.
Referenced by encodeERC20Transfer().

6.7.1.5 CARD_PIN

```
#define CARD_PIN "<CARD_PIN>" /* 4-9 digit PIN, e.g.  "000000000" */
```

Examples

UsdcSigning.ino.

Definition at line 94 of file config.template.h.
Referenced by setup().

6.7.1.6 CARD_PIN_LEN

```
#define CARD_PIN_LEN (9U) /* number of digits in CARD_PIN */
```

Examples

UsdcSigning.ino.

Definition at line 95 of file config.template.h.
Referenced by setup().

6.7.1.7 CHAIN_ID_SEPOLIA

```
#define CHAIN_ID_SEPOLIA 11155111
```

Examples

UsdcSigning.ino.

Definition at line 112 of file config.template.h.
Referenced by setup().

6.7.1.8 GAS_LIMIT_ERC20

```
#define GAS_LIMIT_ERC20 60000ULL
```

Examples

UsdcSigning.ino.

Definition at line 120 of file config.template.h.
Referenced by setup().

6.7.1.9 MAX_FEE

```
#define MAX_FEE 4000000000ULL /* 4 Gwei */
```

Examples

UsdcSigning.ino.

Definition at line 119 of file config.template.h.
Referenced by setup().

6.7.1.10 MAX_PRIORITY_FEE

```
#define MAX_PRIORITY_FEE 2000000000ULL /* 2 Gwei */
```

Examples

UsdcSigning.ino.

Definition at line 118 of file config.template.h.
Referenced by setup().

6.7.1.11 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 five lines under "Option B" and fill in the values. Note: the API Secret must have NO leading or trailing spaces.

Examples

UsdcSigning.ino.

Definition at line 34 of file config.template.h.
Referenced by determineYParity(), fetchNonce(), and sendRawTx().

6.7.1.12 RPC_PATH

```
#define RPC_PATH "/"
```

Examples

UsdcSigning.ino.

Definition at line 36 of file config.template.h.
Referenced by determineYParity(), fetchNonce(), and sendRawTx().

6.7.1.13 RPC_PORT

```
#define RPC_PORT 443
```

Examples

UsdcSigning.ino.

Definition at line 35 of file config.template.h.
Referenced by determineYParity(), fetchNonce(), and sendRawTx().

6.7.1.14 WIFI_CA_CERT

```
#define WIFI_CA_CERT
```

Value:

```
"-----BEGIN CERTIFICATE-----\n" \
"MIIDejCCAmKgAwIBAgIQf+UwvzMTQ77dghYQST2KGzANBgkqhkiG9w0BAQsFADBX\n" \
"MQswCQYDVQQGEwJCRTEZMBcGA1UEChMQR2xvYmFsU2lnbiBudi1zYTEQMA4GA1UE\n" \
"CxMHUm9vdCBDQTEbMBkGA1UEAxMSR2xvYmFsU2lnbiBSb290IENBMB4XDTIzMTEx\n" \
"NTAzNDMyMVoXDTI4MDEyODAwMDA0MlowRzELMAkGA1UEBhMCVVMxIjAgBgNVBAoT\n" \
"GUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBMTEMxFDASBgNVBAMTC0dUUyBSb290IFI0\n" \
"MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAE83Rzp2iLYK5DuDXFgTB7S0md+8Fhzube\n" \
"Rr1r1WEYNa5A3XP3iZEwWus87oV8okB2O6nGuEfYKueSkWpz6bFyOZ8pn6KY019e\n" \
"WIZlD6GEZQbR3IvJx3PIjGov5cSr0R2Ko4H/MIH8MA4GA1UdDwEB/wQEAwIBhjAd\n" \
"BgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIwDwYDVR0TAQH/BAUwAwEB/zAd\n" \
"BgNVHQ4EFgQUgEzW63T/STaj1dj8tT7FavCUHYwwHwYDVR0jBBgwFoAUYHtmGkUN\n" \
"l8qJUC99BM00qP/8/UswNgYIKwYBBQUHAQEEKjAoMCYGCCsGAQUFBzAChhpodHRw\n" \
"Oi8vaS5wa2kuZ29vZy9nc3IxLmNydDAtBgNVHR8EJjAkMCKgIKAehhxodHRwOi8v\n" \
"Yy5wa2kuZ29vZy9yL2dzcjEuY3JsMBMGA1UdIAQMMAowCAYGZ4EMAQIBMA0GCSqG\n" \
"SIb3DQEBCwUAA4IBAQAYQrsPBtYDh5bjP2OBDwmkoWhIDDkic574y04tfzHpn+cJ\n" \
"odI2D4SseesQ6bDrarZ7C30ddLibZatoKiws3UL9xnELz4ct92vID24FfVbiI1hY\n" \
"+SW6FoVHkNeWIP0GCbaM4C6uVdF5dTUsMVs/ZbzNnIdCp5Gxmx5ejvEau8otR/Cs\n" \
"kGN+hr/W5GvT1tMBjgWKZ1i4//emhA1JG1BbPzoLJQvyEotc03lXjTaCzv8mEbep\n" \
"8RqZ7a2CPsgRbuvTPBwcOMBBmuFeU88+FSBX6+7iP0il8b4Z0QFqIwwMHfs/L6K1\n" \
"vepuoxtGzi4CZ68zJpiq1UvSqTbFJjtbD4seiMHl\n" \
"-----END CERTIFICATE-----\n"
```

Examples

UsdcSigning.ino.

Definition at line 60 of file config.template.h.
Referenced by determineYParity(), fetchNonce(), and sendRawTx().

6.7.1.15 WIFI_PASSWORD

```
#define WIFI_PASSWORD "<YOUR_PASSWORD>"
```

Examples

UsdcSigning.ino.

Definition at line 15 of file config.template.h.
Referenced by ensureWiFi(), and setup().

6.7.1.16 WIFI_SSID

```
#define WIFI_SSID "<YOUR_SSID>"
```

Examples

UsdcSigning.ino.

Definition at line 14 of file config.template.h.
Referenced by ensureWiFi(), and setup().

6.8 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_PATH       "/"
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_PATH        "/v3/" RPC_PROJECT_ID                      */
00045 /* #define RPC_API_SECRET  "<YOUR_INFURA_API_SECRET>"                 */
00046
00047 /* --- TLS server certificate pinning (M-04) ----------------------
00048  * The sketch defaults to ISRG Root X1 (Let's Encrypt). If your provider
00049  * uses a different root CA, override WIFI_CA_CERT below with the
00050  * appropriate root in PEM form.
00051  *
00052  * To retrieve the real CA chain of your endpoint:
00053  *   openssl s_client -showcerts -servername HOST -connect HOST:443 </dev/null
00054  *
```

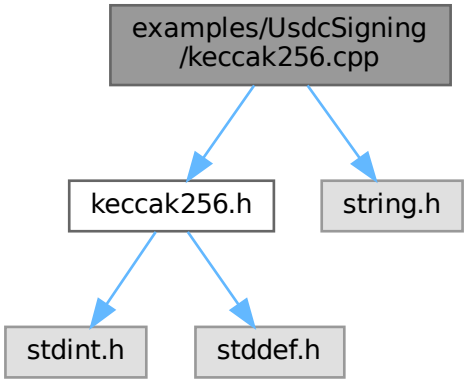
```
00055  * To DISABLE pinning temporarily for development (e.g. to confirm a TLS
00056  * handshake failure is a cert issue, not WiFi), uncomment WIFI_DISABLE_-
00057  * CA_PINNING. The sketch prints a loud warning at boot when set.
00058  *  Never ship firmware with WIFI_DISABLE_CA_PINNING defined -- the
00059  * connection becomes trivially MITM-able. */
00060 #define WIFI_CA_CERT \
00061 "-----BEGIN CERTIFICATE-----\n" \
00062 "MIIDejCCAmKgAwIBAgIQf+UwvzMTQ77dghYQST2KGzANBgkqhkiG9w0BAQsFADBX\n" \
00063 "MQswCQYDVQQGEwJCRTEZMBcGA1UEChMQR2xvYmFsU2lnbiBudi1zYTEQMA4GA1UE\n" \
00064 "CxMHUm9vdCBDQTEbMBkGA1UEAxMSR2xvYmFsU2lnbiBSb290IENBMB4XDTIzMTEx\n" \
00065 "NTAzNDMyMVoXDTI4MDEyODAwMDA0MlowRzELMAkGA1UEBhMCVVMxIjAgBgNVBAoT\n" \
00066 "GUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBMTEMxFDASBgNVBAMTC0dUUyBSb290IFI0\n" \
00067 "MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAE83Rzp2iLYK5DuDXFgTB7S0md+8Fhzube\n" \
00068 "Rr1r1WEYNa5A3XP3iZEwWus87oV8okB2O6nGuEfYKueSkWpz6bFyOZ8pn6KY019e\n" \
00069 "WIZlD6GEZQbR3IvJx3PIjGov5cSr0R2Ko4H/MIH8MA4GA1UdDwEB/wQEAwIBhjAd\n" \
00070 "BgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIwDwYDVR0TAQH/BAUwAwEB/zAd\n" \
00071 "BgNVHQ4EFgQUgEzW63T/STaj1dj8tT7FavCUHYwwHwYDVR0jBBgwFoAUYHtmGkUN\n" \
00072 "l8qJUC99BM00qP/8/UswNgYIKwYBBQUHAQEEKjAoMCYGCCsGAQUFBzAChhpodHRw\n" \
00073 "Oi8vaS5wa2kuZ29vZy9nc3IxLmNydDAtBgNVHR8EJjAkMCKgIKAehhxodHRwOi8v\n" \
00074 "Yy5wa2kuZ29vZy9yL2dzcjEuY3JsMBMGA1UdIAQMMAowCAYGZ4EMAQIBMA0GCSqG\n" \
00075 "SIb3DQEBCwUAA4IBAQAYQrsPBtYDh5bjP2OBDwmkoWhIDDkic574y04tfzHpn+cJ\n" \
00076 "odI2D4SseesQ6bDrarZ7C30ddLibZatoKiws3UL9xnELz4ct92vID24FfVbiI1hY\n" \
00077 "+SW6FoVHkNeWIP0GCbaM4C6uVdF5dTUsMVs/ZbzNnIdCp5Gxmx5ejvEau8otR/Cs\n" \
00078 "kGN+hr/W5GvT1tMBjgWKZ1i4//emhA1JG1BbPzoLJQvyEotc03lXjTaCzv8mEbep\n" \
00079 "8RqZ7a2CPsgRbuvTPBwcOMBBmuFeU88+FSBX6+7iP0il8b4Z0QFqIwwMHfs/L6K1\n" \
00080 "vepuoxtGzi4CZ68zJpiq1UvSqTbFJjtbD4seiMHl\n" \
00081 "-----END CERTIFICATE-----\n"
00082 /* #define WIFI_DISABLE_CA_PINNING */  /* DEV ONLY -- MITM vulnerable */
00083
00084 /* ========================
00085  * Wallet / Keys (SENSITIVE)
00086  * ======================== */
00087 /*  NEVER COMMIT config.h -- it contains credentials.
00088  *
00089  * L-04 -- A hardcoded PIN sits in flash (.rodata) and is recoverable via
00090  * SWD/JTAG. OK for a demo, NOT for production: in prod replace with a
00091  * keypad / BLE prompt / companion chip, then secure_wipe() the buffer
00092  * right after wallet.sign() / wallet.verifyPin(). */
00093
00094 #define CARD_PIN      "<CARD_PIN>"   /* 4-9 digit PIN, e.g. "000000000" */
00095 #define CARD_PIN_LEN  (9U)           /* number of digits in CARD_PIN     */
00096
00097 /* ========================
00098  * Ethereum Addresses
00099  * ======================== */
00100 /* Sender address -- lowercase hex, no 0x prefix */
00101 #define ADDR_FROM         "<SENDER_ADDRESS>"
00102
00103 /* Recipient address */
00104 #define ADDR_TO           "<RECIPIENT_ADDRESS>"
00105
00106 /* USDC contract address (Sepolia testnet) */
00107 #define ADDR_USDC         "<USDC_CONTRACT_ADDRESS>"
00108
00109 /* ========================
00110  * Transaction Parameters
00111  * ======================== */
00112 #define CHAIN_ID_SEPOLIA  11155111
00113
00114 /* Amount in token smallest unit (USDC has 6 decimals) */
00115 #define AMOUNT_USDC       1000000UL   /* 1.0 USDC */
00116
00117 /* Gas parameters (in wei) */
00118 #define MAX_PRIORITY_FEE  2000000000ULL  /* 2 Gwei */
00119 #define MAX_FEE           4000000000ULL  /* 4 Gwei */
00120 #define GAS_LIMIT_ERC20   60000ULL
00121
00122 #endif /* CONFIG_H */
```

6.9 examples/UsdcSigning/keccak256.cpp File Reference

Keccak-256 (SHA3 variant) hash implementation for Ethereum.

```
#include "keccak256.h"
#include <string.h>
```

Include dependency graph for keccak256.cpp:



Macros

- #define KECCAK_ROUNDS 24

Functions

- static uint64_t rol (uint64_t x, int s)

  Rotate a 64-bit integer left by s bits.
- static void keccakf (uint64_t st[25])
- void keccak256 (const uint8_t *in, size_t inlen, uint8_t out[32])

  Compute Keccak-256 hash of input data.

Variables

- static const uint64_t keccakf_rndc [24]
- static const int8_t keccakf_rotc [24]

  Keccak-f[1600] permutation on the state array.
- static const int8_t keccakf_piln [24]

6.9.1 Detailed Description

Keccak-256 (SHA3 variant) hash implementation for Ethereum.
Implements Keccak-256 hashing suitable for Ethereum-style hashing, producing a 32-byte digest. This file contains the internal permutation function and the main keccak256 API.
Definition in file keccak256.cpp.

6.9.2 Macro Definition Documentation

6.9.2.1 KECCAK_ROUNDS

```
#define KECCAK_ROUNDS 24
```

Definition at line 19 of file keccak256.cpp.
Referenced by keccakf().

6.9.3 Function Documentation

6.9.3.1 keccak256()

```
void keccak256 (
            const uint8_t * in,
            size_t inlen,
            uint8_t out[32])
```

Compute Keccak-256 hash of input data.
Compute the Keccak-256 hash of a given input buffer.

Parameters

in	*in*	Pointer to input buffer.
in	*inlen*	Length of input buffer in bytes.
out	*out*	Pointer to a 32-byte buffer to store the hash.

Note

The output buffer must be at least 32 bytes long.

This is Ethereum's Keccak-256 (pre-SHA3 standard padding).

Examples

UsdcSigning.ino.

Definition at line 111 of file keccak256.cpp.
References keccakf().
Referenced by setup().

6.9.3.2 keccakf()

```
void keccakf (
            uint64_t st[25])  [static]
```

Definition at line 63 of file keccak256.cpp.
References KECCAK_ROUNDS, keccakf_piln, keccakf_rndc, keccakf_rotc, and rol().
Referenced by keccak256().

6.9.3.3 rol()

```
uint64_t rol (
            uint64_t x,
            int s)  [inline], [static]
```

Rotate a 64-bit integer left by s bits.

Parameters

x	Value to rotate.
s	Number of bits to rotate.

Returns

Rotated value.

Definition at line 44 of file keccak256.cpp.
Referenced by keccakf().

6.9.4 Variable Documentation

6.9.4.1 keccakf_piln

```
const int8_t keccakf_piln[24]  [static]
```

Initial value:

```
= {
  10, 7, 11, 17, 18, 3, 5, 16, 8, 21, 24, 4,
  15, 23, 19, 13, 12, 2, 20, 14, 22, 9, 6, 1
}
```

Definition at line 58 of file keccak256.cpp.
Referenced by keccakf().

6.9.4.2 keccakf_rndc

```
const uint64_t keccakf_rndc[24]  [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 22 of file keccak256.cpp.
Referenced by keccakf().

6.9.4.3 keccakf_rotc

```
const int8_t keccakf_rotc[24]  [static]
```

Initial value:

```
= {
  1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 2, 14,
  27, 41, 56, 8, 25, 43, 62, 18, 39, 61, 20, 44
}
```

Keccak-f[1600] permutation on the state array.

Parameters

st	25-word state array (1600 bits).

This function applies the 24-round Keccak-f permutation on the input state array in-place.
Definition at line 54 of file keccak256.cpp.
Referenced by keccakf().

6.10 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
00014
00015 #include "keccak256.h"
00016 #include <string.h>
00017
00018 // Number of rounds for the Keccak-f[1600] permutation
00019 #define KECCAK_ROUNDS 24
00020
```

```
00021 // Round constants for Keccak-f[1600]
00022 static const uint64_t keccakf_rndc[24] = {
00023   0x0000000000000001ULL, 0x0000000000008082ULL,
00024   0x800000000000808aULL, 0x8000000080008000ULL,
00025   0x000000000000808bULL, 0x0000000080000001ULL,
00026   0x8000000080008081ULL, 0x8000000000008009ULL,
00027   0x000000000000008aULL, 0x0000000000000088ULL,
00028   0x0000000080008009ULL, 0x000000008000000aULL,
00029   0x000000008000808bULL, 0x800000000000008bULL,
00030   0x8000000000008089ULL, 0x8000000000008003ULL,
00031   0x8000000000008002ULL, 0x8000000000000080ULL,
00032   0x000000000000800aULL, 0x800000008000000aULL,
00033   0x8000000080008081ULL, 0x8000000000008080ULL,
00034   0x0000000080000001ULL, 0x8000000080008008ULL
00035 };
00036
00044 static inline uint64_t rol(uint64_t x, int s) { return (x « s) | (x » (64 - s)); }
00045
00054 static const int8_t keccakf_rotc[24] = {
00055   1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 2, 14,
00056   27, 41, 56, 8, 25, 43, 62, 18, 39, 61, 20, 44
00057 };
00058 static const int8_t keccakf_piln[24] = {
00059   10, 7, 11, 17, 18, 3, 5, 16, 8, 21, 24, 4,
00060   15, 23, 19, 13, 12, 2, 20, 14, 22, 9, 6, 1
00061 };
00062
00063 static void keccakf(uint64_t st[25]) {
00064   for (int r = 0; r < KECCAK_ROUNDS; ++r) {
00065     uint64_t bc[5];
00066     for (int i = 0; i < 5; ++i)
00067       bc[i] = st[i] ^ st[i + 5] ^ st[i + 10] ^ st[i + 15] ^ st[i + 20];
00068
00069     for (int i = 0; i < 5; ++i) {
00070       uint64_t t = bc[(i + 4) % 5] ^ rol(bc[(i + 1) % 5], 1);
00071       for (int j = 0; j < 25; j += 5)
00072         st[j + i] ^= t;
00073     }
00074
00075     uint64_t t = st[1];
00076     for (int i = 0; i < 24; ++i) {
00077       int8_t j = keccakf_piln[i];
00078       uint64_t tmp = st[j];
00079       st[j] = rol(t, keccakf_rotc[i]);
00080       t = tmp;
00081     }
00082
00083     for (int j = 0; j < 25; j += 5) {
00084       uint64_t bc0 = st[j];
00085       uint64_t bc1 = st[j+1];
00086       uint64_t bc2 = st[j+2];
00087       uint64_t bc3 = st[j+3];
00088       uint64_t bc4 = st[j+4];
00089
00090       st[j]   ^= (~bc1) & bc2;
00091       st[j+1] ^= (~bc2) & bc3;
00092       st[j+2] ^= (~bc3) & bc4;
00093       st[j+3] ^= (~bc4) & bc0;
00094       st[j+4] ^= (~bc0) & bc1;
00095     }
00096     st[0] ^= keccakf_rndc[r];
00097   }
00098 }
00099
00110 // cppcheck-suppress unusedFunction
00111 void keccak256(const uint8_t *in, size_t inlen, uint8_t out[32]) {
00112   uint64_t st[25];
00113   memset(st, 0, sizeof(st));
00114
00115   size_t rate = 1088/8; // SHA3-256 rate in bytes
00116
00117   // Absorb full rate blocks
00118   while (inlen >= rate) {
00119     for (size_t i = 0; i < rate/8; ++i) {
00120       uint64_t t = 0;
00121       memcpy(&t, in + i*8, 8);
00122       st[i] ^= t;
00123     }
00124     keccakf(st);
00125     inlen -= rate;
00126     in += rate;
00127   }
00128
00129   // Absorb remaining bytes and pad
00130   uint8_t temp[200];
00131   memset(temp, 0, sizeof(temp));
00132   memcpy(temp, in, inlen);
```

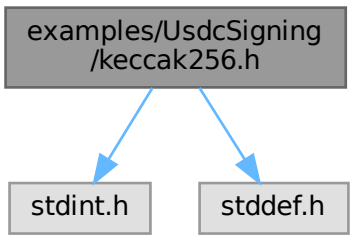
```
00133   temp[inlen] = 0x01;       // Ethereum Keccak-256 padding
00134   temp[rate-1] |= 0x80;
00135
00136   for (size_t i = 0; i < rate/8; ++i) {
00137     uint64_t t = 0;
00138     memcpy(&t, temp + i*8, 8);
00139     st[i] ^= t;
00140   }
00141   keccakf(st);
00142
00143   // Copy first 32 bytes of state as hash output
00144   memcpy(out, st, 32);
00145 }
```

6.11 examples/UsdcSigning/keccak256.h File Reference

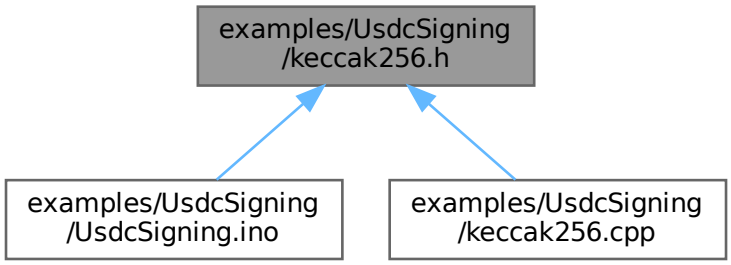
Keccak-256 (SHA3 variant) hash function for Ethereum.

```
#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 *in, size_t inlen, uint8_t out[32])

  Compute the Keccak-256 hash of a given input buffer.

6.11.1 Detailed Description

Keccak-256 (SHA3 variant) hash function for Ethereum.
Provides a function to compute the Keccak-256 hash of an input buffer, producing a 32-byte digest.

Note

This implementation is suitable for Ethereum-style hashing.

Definition in file keccak256.h.

6.11.2 Function Documentation

6.11.2.1 keccak256()

```
void keccak256 (
            const uint8_t * in,
            size_t inlen,
            uint8_t out[32])
```

Compute the Keccak-256 hash of a given input buffer.

Parameters

in	*in*	Pointer to input data.
in	*inlen*	Length of input data in bytes.
out	*out*	Pointer to a 32-byte buffer where the hash will be stored.

Note

The output buffer must be at least 32 bytes long.

This function implements the Ethereum variant of Keccak-256.

Compute the Keccak-256 hash of a given input buffer.

Parameters

in	*in*	Pointer to input buffer.
in	*inlen*	Length of input buffer in bytes.
out	*out*	Pointer to a 32-byte buffer to store the hash.

Note

The output buffer must be at least 32 bytes long.

This is Ethereum's Keccak-256 (pre-SHA3 standard padding).

Definition at line 111 of file keccak256.cpp.
References keccakf().
Referenced by setup().

6.12 keccak256.h

Go to the documentation of this file.

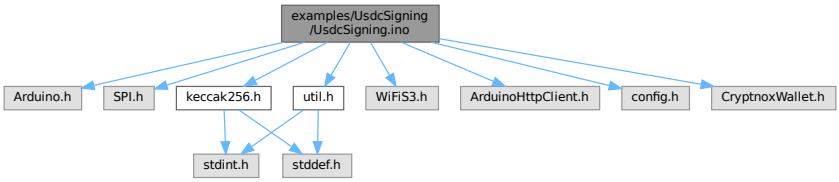
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
```

```
00015
00016 #ifndef KECCAK256_H
00017 #define KECCAK256_H
00018
00019 #include <stdint.h>
00020 #include <stddef.h>
00021
00032 void keccak256(const uint8_t *in, size_t inlen, uint8_t out[32]);
00033
00034 #endif /* KECCAK256_H */
```

6.13 examples/UsdcSigning/UsdcSigning.ino File Reference

```
#include <Arduino.h>
#include <SPI.h>
#include "keccak256.h"
#include <WiFiS3.h>
#include <ArduinoHttpClient.h>
#include "util.h"
#include "config.h"
#include <CryptnoxWallet.h>
```

Include dependency graph for UsdcSigning.ino:



Classes

- struct Tx2

  Ethereum EIP-1559 transaction structure.

Macros

- #define PN532_SS_PIN (10U)

  PN532 SPI slave-select pin.
- #define CARD_PIN "000000000"
- #define CARD_PIN_LEN (9U)
- #define RPC_PATH "/"
- #define WIFI_CA_CERT
- #define ERC20_TRANSFER_SEL_0 0xa9U
- #define ERC20_TRANSFER_SEL_1 0x05U
- #define ERC20_TRANSFER_SEL_2 0x9cU
- #define ERC20_TRANSFER_SEL_3 0xbbU
- #define ERC20_INDEX_OFFSET 64U
- #define YPARITY_UNKNOWN 0xFFU

  Sentinel returned by determineYParity() when recovery fails.
- #define HTTP_OK 200

  Expected HTTP 200 OK status code.
- #define TX_MAX_RETRIES 3U

  Maximum number of send-transaction attempts before giving up.

- #define TX_RETRY_DELAY_MS 2000U

  Delay in ms between send-transaction retry attempts.
- #define WIFI_RETRY_MAX 20U

  Maximum WiFi reconnect poll iterations (each iteration waits 500 ms).
- #define HEX_CHAR_BUF_SIZE 3U

  Buffer size for a two-hex-char + NUL string used in byte-to-hex conversion.
- #define ECRECOVER_V_PAD_CHARS 62U

  Number of leading zero hex characters in the ecrecover v-field padding.
- #define ECRECOVER_V_BASE 27U

  Base value for Ethereum ecrecover v parameter (yParity=0 → v=27, yParity=1 → v=28).
- #define RLP_ITEM_OR_FAIL(BUF, CAP, OFF, IN, IN_LEN)

Functions

- PN532Adapter nfc (serialAdapter, PN532_SS_PIN, &SPI)
- static void printHex (const char *label, const uint8_t *data, size_t len)
- static bool ensureWiFi ()
- static size_t rlpEncodeTxBody (uint8_t *buf, size_t bufCap, const Tx2 &tx)
- static size_t rlpFinalize (uint8_t *out, size_t outCap, const uint8_t *buf, size_t off)
- size_t rlpEncodeUnsignedTx (const Tx2 &tx, uint8_t *out, size_t outCap)
- size_t rlpEncodeSignedTx (const Tx2 &tx, const uint8_t *r, const uint8_t *s, const uint8_t *v, uint8_t *out, size_t outCap)
- size_t encodeERC20Transfer (uint8_t *out)

  Encode calldata for ERC-20 transfer(address to, uint256 amount).
- bool sendRawTx (const uint8_t *raw, size_t len)

  Send a raw signed transaction via JSON-RPC.
- uint8_t determineYParity (const uint8_t *hash, const uint8_t *r, const uint8_t *s)

  Determine EIP-1559 yParity by calling the Ethereum ecrecover precompile.
- uint8_t fetchNonce (uint64_t *nonce)

  Fetch the current nonce for ADDR_FROM via eth_getTransactionCount.
- void setup ()

  Arduino setup: init PN532, connect WiFi, build tx, sign with Cryptnox card, send.
- void loop ()

  Arduino loop (empty — transaction is sent once in setup).

Variables

- ArduinoLoggerAdapter serialAdapter
- ArduinoCryptoProvider cryptoProvider
- ArduinoPlatform platform
- CryptnoxWallet wallet (nfc, serialAdapter, cryptoProvider, platform)
- static const char hexChars [ ] = "0123456789abcdef"

6.13.1 Macro Definition Documentation

6.13.1.1 CARD_PIN

```
#define CARD_PIN "000000000"
```

Definition at line 37 of file UsdcSigning.ino.

6.13.1.2 CARD_PIN_LEN

```
#define CARD_PIN_LEN (9U)
```

Definition at line 38 of file UsdcSigning.ino.

6.13.1.3 ECRECOVER_V_BASE

```
#define ECRECOVER_V_BASE 27U
```

Base value for Ethereum ecrecover v parameter (yParity=0 → v=27, yParity=1 → v=28).

Examples

UsdcSigning.ino.

Definition at line 128 of file UsdcSigning.ino.
Referenced by determineYParity().

6.13.1.4 ECRECOVER_V_PAD_CHARS

```
#define ECRECOVER_V_PAD_CHARS 62U
```

Number of leading zero hex characters in the ecrecover v-field padding.

Examples

UsdcSigning.ino.

Definition at line 126 of file UsdcSigning.ino.
Referenced by determineYParity().

6.13.1.5 ERC20_INDEX_OFFSET

```
#define ERC20_INDEX_OFFSET 64U
```

Examples

UsdcSigning.ino.

Definition at line 109 of file UsdcSigning.ino.
Referenced by encodeERC20Transfer().

6.13.1.6 ERC20_TRANSFER_SEL_0

```
#define ERC20_TRANSFER_SEL_0 0xa9U
```

ERC-20 transfer(address,uint256) function selector: keccak256("transfer(address,uint256)")[0..3]

Examples

UsdcSigning.ino.

Definition at line 104 of file UsdcSigning.ino.
Referenced by encodeERC20Transfer().

6.13.1.7 ERC20_TRANSFER_SEL_1

```
#define ERC20_TRANSFER_SEL_1 0x05U
```

Examples

UsdcSigning.ino.

Definition at line 105 of file UsdcSigning.ino.
Referenced by encodeERC20Transfer().

6.13.1.8 ERC20_TRANSFER_SEL_2

```
#define ERC20_TRANSFER_SEL_2 0x9cU
```

Examples

UsdcSigning.ino.

Definition at line 106 of file UsdcSigning.ino.
Referenced by encodeERC20Transfer().

6.13.1.9 ERC20_TRANSFER_SEL_3

```
#define ERC20_TRANSFER_SEL_3 0xbbU
```

Examples

UsdcSigning.ino.

Definition at line 107 of file UsdcSigning.ino.
Referenced by encodeERC20Transfer().

6.13.1.10 HEX_CHAR_BUF_SIZE

```
#define HEX_CHAR_BUF_SIZE 3U
```

Buffer size for a two-hex-char + NUL string used in byte-to-hex conversion.

Examples

UsdcSigning.ino.

Definition at line 124 of file UsdcSigning.ino.
Referenced by sendRawTx().

6.13.1.11 HTTP_OK

```
#define HTTP_OK 200
```

Expected HTTP 200 OK status code.

Examples

UsdcSigning.ino.

Definition at line 115 of file UsdcSigning.ino.
Referenced by determineYParity(), fetchNonce(), and sendRawTx().

6.13.1.12 PN532_SS_PIN

```
#define PN532_SS_PIN (10U)
```

PN532 SPI slave-select pin.
Definition at line 31 of file UsdcSigning.ino.

6.13.1.13 RLP_ITEM_OR_FAIL

```
#define RLP_ITEM_OR_FAIL(
              BUF,
              CAP,
              OFF,
              IN,
              IN_LEN)
```

Value:

```
    do {                                                                \
        uint32_t _w = RlpEncodeItem((BUF) + (OFF), (CAP) - (OFF),       \
                                    (IN), (IN_LEN));                    \
        if (_w == 0U) { return 0U; }                                    \
        (OFF) += _w;                                                    \
    } while (0)
```

Examples

UsdcSigning.ino.

Definition at line 252 of file UsdcSigning.ino.
Referenced by rlpEncodeTxBody().

6.13.1.14 RPC_PATH

```
#define RPC_PATH "/"
```
Definition at line 44 of file UsdcSigning.ino.

6.13.1.15 TX_MAX_RETRIES

```
#define TX_MAX_RETRIES 3U
```
Maximum number of send-transaction attempts before giving up.

Examples

UsdcSigning.ino.

Definition at line 118 of file UsdcSigning.ino.
Referenced by fetchNonce(), and sendRawTx().

6.13.1.16 TX_RETRY_DELAY_MS

```
#define TX_RETRY_DELAY_MS 2000U
```
Delay in ms between send-transaction retry attempts.

Examples

UsdcSigning.ino.

Definition at line 120 of file UsdcSigning.ino.
Referenced by sendRawTx().

6.13.1.17 WIFI_CA_CERT

```
#define WIFI_CA_CERT
```
Definition at line 59 of file UsdcSigning.ino.

6.13.1.18 WIFI_RETRY_MAX

```
#define WIFI_RETRY_MAX 20U
```
Maximum WiFi reconnect poll iterations (each iteration waits 500 ms).

Examples

UsdcSigning.ino.

Definition at line 122 of file UsdcSigning.ino.
Referenced by ensureWiFi().

6.13.1.19 YPARITY_UNKNOWN

```
#define YPARITY_UNKNOWN 0xFFU
```
Sentinel returned by determineYParity() when recovery fails.

Examples

UsdcSigning.ino.

Definition at line 112 of file UsdcSigning.ino.
Referenced by determineYParity(), and setup().

6.13.2 Function Documentation

6.13.2.1 determineYParity()

```
uint8_t determineYParity (
            const uint8_t * hash,
            const uint8_t * r,
            const uint8_t * s)
```

Determine EIP-1559 yParity by calling the Ethereum ecrecover precompile.
Tries v=27 (yParity=0) and v=28 (yParity=1). Compares the recovered address with ADDR_FROM (defined in config.h) to pick the correct value.

Parameters

hash	Keccak-256 transaction hash (32 bytes).
r	Signature r component (32 bytes).
s	Signature s component (32 bytes).

Returns

0 or 1 on success, YPARITY_UNKNOWN on failure.

Examples

UsdcSigning.ino.

Definition at line 471 of file UsdcSigning.ino.
References ADDR_FROM, ECRECOVER_V_BASE, ECRECOVER_V_PAD_CHARS, ensureWiFi(), F, hexChars, HTTP_OK, RPC_HOST, RPC_PATH, RPC_PORT, WIFI_CA_CERT, and YPARITY_UNKNOWN.
Referenced by setup().

6.13.2.2 encodeERC20Transfer()

```
size_t encodeERC20Transfer (
            uint8_t * out)
```

Encode calldata for ERC-20 transfer(address to, uint256 amount).

Parameters

out	Output buffer (at least 68 bytes)

Returns

Calldata length (always 68)

Examples

UsdcSigning.ino.

Definition at line 346 of file UsdcSigning.ino.
References ADDR_TO, AMOUNT_USDC, ERC20_INDEX_OFFSET, ERC20_TRANSFER_SEL_0, ERC20_TRANSFER_SEL_1, ERC20_TRANSFER_SEL_2, ERC20_TRANSFER_SEL_3, F, hexToBytes(), and CW_Utils::secure_wipe().
Referenced by setup().

6.13.2.3 ensureWiFi()

```
bool ensureWiFi ()  [static]
```

Examples

UsdcSigning.ino.

Definition at line 239 of file UsdcSigning.ino.
References WIFI_PASSWORD, WIFI_RETRY_MAX, and WIFI_SSID.
Referenced by determineYParity(), fetchNonce(), and sendRawTx().

6.13.2.4 fetchNonce()

```
uint8_t fetchNonce (
            uint64_t * nonce)
```

Fetch the current nonce for ADDR_FROM via eth_getTransactionCount.

Parameters

nonce	Output: nonce value on success (note: 0 is a valid nonce for a fresh address).

Returns

0 on success, 1 on failure.

Examples

UsdcSigning.ino.

Definition at line 586 of file UsdcSigning.ino.
References ADDR_FROM, ensureWiFi(), F, fromHex(), HTTP_OK, RPC_HOST, RPC_PATH, RPC_PORT, TX_MAX_RETRIES, and WIFI_CA_CERT.
Referenced by setup().

6.13.2.5 loop()

```
void loop ()
```

Arduino loop (empty — transaction is sent once in setup).
Definition at line 840 of file UsdcSigning.ino.

6.13.2.6 nfc()

```
PN532Adapter nfc (
            serialAdapter ,
            PN532_SS_PIN ,
            & SPI)
```

References PN532_SS_PIN, and serialAdapter.

6.13.2.7 printHex()

```
void printHex (
            const char * label,
            const uint8_t * data,
            size_t len)  [static]
```

Examples

UsdcSigning.ino.

Definition at line 147 of file UsdcSigning.ino.
References F, and hexChars.
Referenced by setup().

6.13.2.8 rlpEncodeSignedTx()

```
size_t rlpEncodeSignedTx (
            const Tx2 & tx,
            const uint8_t * r,
            const uint8_t * s,
            const uint8_t * v,
            uint8_t * out,
            size_t outCap)
```

Examples

UsdcSigning.ino.

Definition at line 311 of file UsdcSigning.ino.
References RlpEncodeItem(), rlpEncodeTxBody(), rlpFinalize(), CW_Utils::secure_wipe(), and trimLeadingZeros().
Referenced by setup().

6.13.2.9 rlpEncodeTxBody()

```
size_t rlpEncodeTxBody (
            uint8_t * buf,
            size_t bufCap,
            const Tx2 & tx)  [static]
```

Examples

UsdcSigning.ino.

Definition at line 260 of file UsdcSigning.ino.
References Tx2::chainId, ConvertNumberToUintArray(), Tx2::data, Tx2::dataLen, F, Tx2::gasLimit, hexToBytes(), Tx2::maxFeePerGas, Tx2::maxPriorityFeePerGas, Tx2::nonce, RLP_ITEM_OR_FAIL, Tx2::to, and Tx2::value.
Referenced by rlpEncodeSignedTx(), and rlpEncodeUnsignedTx().

6.13.2.10 rlpEncodeUnsignedTx()

```
size_t rlpEncodeUnsignedTx (
            const Tx2 & tx,
            uint8_t * out,
            size_t outCap)
```

Examples

UsdcSigning.ino.

Definition at line 304 of file UsdcSigning.ino.
References rlpEncodeTxBody(), and rlpFinalize().
Referenced by setup().

6.13.2.11 rlpFinalize()

```
size_t rlpFinalize (
            uint8_t * out,
            size_t outCap,
            const uint8_t * buf,
            size_t off)  [static]
```

Examples

UsdcSigning.ino.

Definition at line 288 of file UsdcSigning.ino.
References RlpEncodeWholeHeader(), and CW_Utils::safe_memcpy().
Referenced by rlpEncodeSignedTx(), and rlpEncodeUnsignedTx().

6.13.2.12 sendRawTx()

```
bool sendRawTx (
            const uint8_t * raw,
            size_t len)
```

Send a raw signed transaction via JSON-RPC.

Parameters

raw	Signed transaction bytes
len	Length of raw transaction in bytes

Returns

true if the RPC call succeeded without an error field, false otherwise.

Examples

UsdcSigning.ino.

Definition at line 370 of file UsdcSigning.ino.
References ensureWiFi(), F, HEX_CHAR_BUF_SIZE, hexChars, HTTP_OK, RPC_HOST, RPC_PATH, RPC_PORT, TX_MAX_RETRIES, TX_RETRY_DELAY_MS, and WIFI_CA_CERT.
Referenced by setup().

6.13.2.13 setup()

```
void setup ()
```

Arduino setup: init PN532, connect WiFi, build tx, sign with Cryptnox card, send.
Definition at line 672 of file UsdcSigning.ino.
References ADDR_USDC, CARD_PIN, CARD_PIN_LEN, CHAIN_ID_SEPOLIA, Tx2::chainId, CW_HASH_SIZE, CW_OK, CW_SIGN_DERIVE_K1, CW_SIGN_NO_KEY_LOADED, CW_SIGN_PIN_INCORRECT, CW_SIGN_SIG_ECDSA_LOW_S, CW_SIGN_WITH_PIN, Tx2::data, Tx2::dataLen, CW_SignRequest::derivePath, CW_SignRequest::derivePathLength, determineYParity(), encodeERC20Transfer(), CW_SignResult::errorCode, F, fetchNonce(), GAS_LIMIT_ERC20, Tx2::gasLimit, CW_SignRequest::hash, CW_SignRequest::hashLength, HEX, keccak256(), MAX_FEE, MAX_PRIORITY_FEE, Tx2::maxFeePerGas, Tx2::maxPriorityFeePerGas, Tx2::nonce, CW_SignRequest::pin, printHex(), rlpEncodeSignedTx(), rlpEncodeUnsignedTx(), CW_Utils::safe_memcpy(), CW_Utils::secure_wipe(), sendRawTx(), CW_SignResult::signature, Tx2::to, Tx2::value, wallet, WIFI_PASSWORD, WIFI_SSID, and YPARITY_UNKNOWN.

6.13.3 Variable Documentation

6.13.3.1 cryptoProvider

```
ArduinoCryptoProvider cryptoProvider
```

Definition at line 98 of file UsdcSigning.ino.

6.13.3.2 hexChars

```
const char hexChars[] = "0123456789abcdef"  [static]
```

Examples

UsdcSigning.ino.

Definition at line 145 of file UsdcSigning.ino.
Referenced by determineYParity(), printHex(), and sendRawTx().

6.13.3.3 platform

```
ArduinoPlatform platform
```

Definition at line 99 of file UsdcSigning.ino.

6.13.3.4 serialAdapter

```
ArduinoLoggerAdapter serialAdapter
```

Definition at line 97 of file UsdcSigning.ino.

6.13.3.5 wallet

```
CryptnoxWallet wallet(nfc, serialAdapter, cryptoProvider, platform) (
             nfc ,
             serialAdapter ,
             cryptoProvider ,
             platform )
```

6.14 UsdcSigning.ino

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
00020
00021 #include <Arduino.h>
00022 #include <SPI.h>
00023 #include "keccak256.h"
00024 #include <WiFiS3.h>
00025 #include <ArduinoHttpClient.h>
00026 #include "util.h"
00027 #include "config.h"
00028 #include <CryptnoxWallet.h>
00029
00031 #define PN532_SS_PIN  (10U)
00032
00033 /* Fallback PIN -- define CARD_PIN and CARD_PIN_LEN in config.h to override.
00034  * L-04: hardcoded PIN lives in flash, recoverable via SWD/JTAG -- OK for
00035  * demo, not for prod. See config.template.h for safer patterns. */
00036 #ifndef CARD_PIN
00037 #  define CARD_PIN     "000000000"
00038 #  define CARD_PIN_LEN (9U)
00039 #endif
00040
00041 /* Default RPC path -- PublicNode accepts JSON-RPC at root.
00042  * Infura requires /v3/{PROJECT_ID}: define RPC_PATH in config.h for that case. */
00043 #ifndef RPC_PATH
00044 #  define RPC_PATH "/"
00045 #endif
00046
00047 /* M-04: TLS server-certificate pinning.
00048  *
00049  * Without setCACert(), WiFiSSLClient accepts ANY certificate the RPC endpoint
00050  * presents -- a network attacker can MITM the connection and feed crafted
00051  * nonces / gas prices to make the device sign incorrect transactions, or
00052  * exfiltrate the basic-auth credentials sent to Infura.
00053  *
00054  * The default below is ISRG Root X1 (Let's Encrypt), which signs the chains
00055  * used by PublicNode (the template's default provider). For a different
00056  * provider (Infura/DigiCert, Cloudflare, etc.), override WIFI_CA_CERT in
00057  * config.h with the appropriate root in PEM form. */
00058 #ifndef WIFI_CA_CERT
00059 #  define WIFI_CA_CERT \
00060 "-----BEGIN CERTIFICATE-----\n" \
00061 "MIIFazCCA1OgAwIBAgIRAIIQz7DSQONZRGPgu2OCiwAwDQYJKoZIhvcNAQELBQAw\n" \
00062 "TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh\n" \
00063 "cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMTUwNjA0MTEwNDM4\n" \
00064 "WhcNMzUwNjA0MTEwNDM4WjBPMQswCQYDVQQGEwJVUzEpMCcGA1UEChMgSW50ZXJu\n" \
00065 "ZXQgU2VjdXJpdHkgUmVzZWFyY2ggR3JvdXAxFTATBgNVBAMTDElTUkcgUm9vdCBY\n" \
00066 "MTCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAK3oJHP0FDfzm54rVygc\n" \
00067 "h77ct984kIxuPOZXoHj3dcKi/vVqbvYATyjb3miGbESTtrFj/RQSa78f0uoxmyF+\n" \
00068 "0TM8ukj13Xnfs7j/EvEhmkvBioZxaUpmZmyPfjxwv60pIgbz5MDmgK7iS4+3mX6U\n" \
00069 "A5/TR5d8mUgjU+g4rk8Kb4Mu0UlXjIB0ttov0DiNewNwIRt18jA8+o+u3dpjq+sW\n" \
00070 "T8KOEUt+zwvo/7V3LvSye0rgTBIlDHCNAymg4VMk7BPZ7hm/ELNKjD+Jo2FR3qyH\n" \
00071 "B5T0Y3HsLuJvW5iB4YlcNHlsdu87kGJ55tukmi8mxdAQ4Q7e2RCOFvu396j3x+UC\n" \
```

```
00072 "B5iPNgiV5+I3lg02dZ77DnKxHZu8A/lJBdiB3QW0KtZB6awBdpUKD9jf1b0SHzUv\n" \
00073 "KBds0pjBqAlkd25HN7rOrFleaJ1/ctaJxQZBKT5ZPt0m9STJEadao0xAH0ahmbWn\n" \
00074 "OlFuhjuefXKnEgV4We0+UXgVCwOPjdAvBbI+e0ocS3MFEvzG6uBQE3xDk3SzynTn\n" \
00075 "jh8BCNAwlFtxNrQHusEwMFxIt4I7mKZ9YIqioymCzLq9gwQbooMDQaHWBfEbwrbw\n" \
00076 "qHyGO0aoSCqI3Haadr8faqU9GY/rOPNk3sgrDQoo//fb4hVC1CLQJ13hef4Y53CI\n" \
00077 "rU7m2Ys6xt0nUW7/vGT1M0NPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV\n" \
00078 "HRMBAf8EBTADAQH/MB0GA1UdDgQWBBR5tFnme7bl5AFzgAiIyBpY9umbbjANBgkq\n" \
00079 "hkiG9w0BAQsFAAOCAgEAVR9YqbyyqFDQDLHYGmkgJykIrGF1XIpu+ILlaS/V9lZL\n" \
00080 "ubhzEFnTIZd+50xx+7LSYK05qAvqFyFWhfFQDlnrzuBZ6brJFe+GnY+EgPbk6ZGQ\n" \
00081 "3BebYhtF8GaV0nxvwuo77x/Py9auJ/GpsMiu/Xl+mvoiBOv/2X/qkSsisRcOj/KK\n" \
00082 "NFtY2PwByVS5uCbMiogziUwthDyC3+6WVwW6LLv3xLfHTjuCvjHIInNzktHCgKQ5\n" \
00083 "ORAzI4JMPJ+GslWYHb4phowim57iaztXOoJwTdwJx4nLCgdNbOhdjsnvzqvHu7Ur\n" \
00084 "TkXWStAmzOVyyghqpZXjFaH3pO3JLF+l+/+sKAIuvtd7u+Nxe5AW0wdeRlN8NwdC\n" \
00085 "jNPElpzVmbUq4JUagEiuTDkHzsxHpFKVK7q4+63SM1N95R1NbdWhscdCb+ZAJzVc\n" \
00086 "oyi3B43njTOQ5yOf+lCceWxG1bQVs5ZufpsMljq4Ui0/1lvh+wjChP4kqKOJ2qxq\n" \
00087 "4RgqsahDYVvTH9w7jXbyLeiNdd8XM2w9U/t7y0Ff/9yi0GE44Za4rF2LN9d11TPA\n" \
00088 "mRGunUHBcnWEvgJBQl9nJEiU0Zsnvgc/ubhPgXRR4Xq37Z0j4r7g1SgEEzwxA57d\n" \
00089 "emyPxgcYxn/eR44/KJ4EBs+lVDR3veyJm+kXQ99b21/+jh5Xos1AnX5iItreGCc=\n" \
00090 "-----END CERTIFICATE-----\n"
00091 #endif
00092
00093 /* L-05 -- PRODUCTION: USB-CDC Serial is readable by any host process.
00094  * Logs leak RPC URL, basic-auth header, nonce, recipient, tx hash, and
00095  * (if CW_DEBUG_LOGGING=1) secure-channel ciphertext + IVs. Before ship:
00096  * swap for `NullLoggerAdapter serialAdapter;` and drop Serial.begin(). */
00097 ArduinoLoggerAdapter  serialAdapter;
00098 ArduinoCryptoProvider cryptoProvider;
00099 ArduinoPlatform       platform;
00100 PN532Adapter          nfc(serialAdapter, PN532_SS_PIN, &SPI);
00101 CryptnoxWallet        wallet(nfc, serialAdapter, cryptoProvider, platform);
00102
00104 #define ERC20_TRANSFER_SEL_0  0xa9U
00105 #define ERC20_TRANSFER_SEL_1  0x05U
00106 #define ERC20_TRANSFER_SEL_2  0x9cU
00107 #define ERC20_TRANSFER_SEL_3  0xbbU
00108
00109 #define ERC20_INDEX_OFFSET 64U
00110
00112 #define YPARITY_UNKNOWN 0xFFU
00113
00115 #define HTTP_OK 200
00116
00118 #define TX_MAX_RETRIES       3U
00120 #define TX_RETRY_DELAY_MS 2000U
00122 #define WIFI_RETRY_MAX      20U
00124 #define HEX_CHAR_BUF_SIZE    3U
00126 #define ECRECOVER_V_PAD_CHARS 62U
00128 #define ECRECOVER_V_BASE    27U
00129
00133 struct Tx2 {
00134     uint64_t nonce;
00135     uint64_t maxPriorityFeePerGas;
00136     uint64_t maxFeePerGas;
00137     uint64_t gasLimit;
00138     const char* to;
00139     uint64_t value;
00140     const uint8_t* data;
00141     size_t dataLen;
00142     uint32_t chainId;
00143 };
00144
00145 static const char hexChars[] = "0123456789abcdef";
00146
00147 static void printHex(const char* label, const uint8_t* data, size_t len) {
00148     Serial.print(label);
00149     Serial.print(F(": 0x"));
00150     char buf[3]; buf[2] = '\0';
00151     for (size_t i = 0; i < len; i++) {
00152         buf[0] = hexChars[data[i] >> 4];
00153         buf[1] = hexChars[data[i] & 0x0f];
00154         Serial.print(buf);
00155     }
00156     Serial.println();
00157 }
00158
00159 #if defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET)
00161 #define AUTH_CRED_BUF_SIZE    128U
00163 #define AUTH_HEADER_BUF_SIZE  200U
00164
00165 /* NEW-3: base64Encode now requires the output capacity and returns false
00166  * if it cannot write the full (ceil(inputLen/3)*4 + 1) bytes. Prevents
00167  * silent stack overflow if AUTH_HEADER_BUF_SIZE is later reduced or a
00168  * caller passes a too-small buffer. */
00169 static bool base64Encode(const char* input, size_t inputLen, char* output, size_t outputCap) {
00170     static const char kAlphabet[] =
00171         "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
00172     const size_t required = (((inputLen + 2U) / 3U) * 4U) + 1U; /* +1 for NUL */
```

```
00173     if (outputCap < required) {
00174         return false;
00175     }
00176     size_t i = 0U;
00177     size_t o = 0U;
00178     while (i < inputLen) {
00179         uint32_t u0 = static_cast<uint32_t>(static_cast<uint8_t>(input[i]));
00180         uint32_t u1 = 0U;
00181         uint32_t u2 = 0U;
00182         if ((i + 1U) < inputLen) {
00183             u1 = static_cast<uint32_t>(static_cast<uint8_t>(input[i + 1U]));
00184         }
00185         if ((i + 2U) < inputLen) {
00186             u2 = static_cast<uint32_t>(static_cast<uint8_t>(input[i + 2U]));
00187         }
00188         output[o] = kAlphabet[static_cast<uint8_t>(u0 » 2U)];
00189         o++;
00190         output[o] = kAlphabet[static_cast<uint8_t>(((u0 & 0x03U) « 4U) | (u1 » 4U))];
00191         o++;
00192         if ((i + 1U) < inputLen) {
00193             output[o] = kAlphabet[static_cast<uint8_t>(((u1 & 0x0FU) « 2U) | (u2 » 6U))];
00194         } else {
00195             output[o] = '=';
00196         }
00197         o++;
00198         if ((i + 2U) < inputLen) {
00199             output[o] = kAlphabet[static_cast<uint8_t>(u2 & 0x3FU)];
00200         } else {
00201             output[o] = '=';
00202         }
00203         o++;
00204         i += 3U;
00205     }
00206     output[o] = '\0';
00207     return true;
00208 }
00209
00210 static void buildBasicAuthHeader(char* buf, size_t bufSize) {
00211     static const char kPrefix[] = "Basic ";
00212     const size_t prefixLen = sizeof(kPrefix) - 1U;
00213     char cred[AUTH_CRED_BUF_SIZE];
00214     // cppcheck-suppress invalidPrintfArgType_s -- macros are char* when defined in config.h; --force
      evaluates this block without knowing their types
00215     int written = snprintf(cred, sizeof(cred), "%s:%s", RPC_PROJECT_ID, RPC_API_SECRET);
00216     /* H-04: snprintf returns the number of bytes it WOULD have written
00217      * (excluding NUL). A value >= sizeof(cred) means the project id +
00218      * secret were truncated -- sending a truncated Authorization header
00219      * causes opaque 401s and could leak credentials in retry traces. */
00220     if ((written < 0) || ((size_t)written >= sizeof(cred))) {
00221         Serial.println(F("[fatal] RPC_PROJECT_ID + RPC_API_SECRET exceed AUTH_CRED_BUF_SIZE"));
00222         while (true) { /* halt -- do not send a truncated basic-auth header */ }
00223     }
00224     /* NEW-1: ensure the output buffer can hold prefix + base64(cred) + NUL.
00225      * base64 expands by ~4/3; the helper rejects undersized buffers. */
00226     if (prefixLen >= bufSize) {
00227         Serial.println(F("[fatal] AUTH_HEADER_BUF_SIZE too small for prefix"));
00228         while (true) {}
00229     }
00230     (void)CW_Utils::safe_memcpy(reinterpret_cast<uint8_t*>(buf), bufSize,
00231                                 reinterpret_cast<const uint8_t*>(kPrefix), prefixLen);
00232     if (!base64Encode(cred, strlen(cred), buf + prefixLen, bufSize - prefixLen)) {
00233         Serial.println(F("[fatal] base64 output exceeds AUTH_HEADER_BUF_SIZE"));
00234         while (true) {}
00235     }
00236 }
00237 #endif /* defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET) */
00238
00239 static bool ensureWiFi() {
00240     if (WiFi.status() == WL_CONNECTED) return true;
00241     WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
00242     uint8_t retries = WIFI_RETRY_MAX;
00243     while ((retries > 0U) && (WiFi.status() != WL_CONNECTED)) {
00244         retries--;
00245         delay(500U);
00246     }
00247     return WiFi.status() == WL_CONNECTED;
00248 }
00249
00250 /* Each RlpEncodeItem call returns 0 on overflow; bail out immediately so
00251  * the caller sees rlpLen == 0 and halts before broadcasting garbage. */
00252 #define RLP_ITEM_OR_FAIL(BUF, CAP, OFF, IN, IN_LEN)                          \
00253     do {                                                                     \
00254         uint32_t _w = RlpEncodeItem((BUF) + (OFF), (CAP) - (OFF),            \
00255                                     (IN), (IN_LEN));                         \
00256         if (_w == 0U) { return 0U; }                                         \
00257         (OFF) += _w;                                                         \
00258     } while (0)
```

```
00259
00260 static size_t rlpEncodeTxBody(uint8_t* buf, size_t bufCap, const Tx2& tx) {
00261     size_t off = 0;
00262     uint8_t tmp[8];
00263     size_t tmpLen;
00264     tmpLen = ConvertNumberToUintArray(tmp, tx.chainId);
00265     RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
00266     tmpLen = ConvertNumberToUintArray(tmp, tx.nonce);
00267     RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
00268     tmpLen = ConvertNumberToUintArray(tmp, tx.maxPriorityFeePerGas);
00269     RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
00270     tmpLen = ConvertNumberToUintArray(tmp, tx.maxFeePerGas);
00271     RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
00272     tmpLen = ConvertNumberToUintArray(tmp, tx.gasLimit);
00273     RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
00274     uint8_t addr[20];
00275     if (!hexToBytes(tx.to, addr, 20)) {
00276         Serial.println(F("[fatal] tx.to is not a valid 40-char hex string"));
00277         while (true) { /* halt to avoid broadcasting a malformed transaction */ }
00278     }
00279     RLP_ITEM_OR_FAIL(buf, bufCap, off, addr, 20U);
00280     tmpLen = ConvertNumberToUintArray(tmp, tx.value);
00281     RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
00282     RLP_ITEM_OR_FAIL(buf, bufCap, off, tx.data, (uint32_t)tx.dataLen);
00283     if (off >= bufCap) { return 0U; }   /* room for the access-list terminator 0xC0 */
00284     buf[off++] = 0xC0;
00285     return off;
00286 }
00287
00288 static size_t rlpFinalize(uint8_t* out, size_t outCap, const uint8_t* buf, size_t off) {
00289     uint8_t header[8];
00290     size_t header_len = RlpEncodeWholeHeader(header, sizeof(header), off);
00291     if (header_len == 0U) { return 0U; }
00292     /* Reject early if the total wouldn't fit (1 type byte + header + body). */
00293     if ((1U + header_len + off) > outCap) {
00294         return 0U;
00295     }
00296     size_t out_off = 0U;
00297     out[out_off++] = 0x02;
00298     (void)CW_Utils::safe_memcpy(out + out_off, outCap - out_off, header, header_len);
00299     out_off += header_len;
00300     (void)CW_Utils::safe_memcpy(out + out_off, outCap - out_off, buf, off);
00301     return out_off + off;
00302 }
00303
00304 size_t rlpEncodeUnsignedTx(const Tx2& tx, uint8_t* out, size_t outCap) {
00305     uint8_t buf[1024];
00306     size_t off = rlpEncodeTxBody(buf, sizeof(buf), tx);
00307     if (off == 0U) { return 0U; }
00308     return rlpFinalize(out, outCap, buf, off);
00309 }
00310
00311 size_t rlpEncodeSignedTx(const Tx2& tx, const uint8_t* r, const uint8_t* s, const uint8_t* v,
00312                          uint8_t* out, size_t outCap) {
00313     uint8_t buf[1024];
00314     size_t off = rlpEncodeTxBody(buf, sizeof(buf), tx);
00315     if (off == 0U) { return 0U; }
00316     uint32_t w;
00317     w = RlpEncodeItem(buf + off, sizeof(buf) - off, v, 1U);
00318     if (w == 0U) { return 0U; }
00319     off += w;
00320     uint8_t tmp_r[32];
00321     size_t tmp_len = trimLeadingZeros(tmp_r, sizeof(tmp_r), r, 32U);
00322     if (tmp_len == 0U) { return 0U; }
00323     w = RlpEncodeItem(buf + off, sizeof(buf) - off, tmp_r, (uint32_t)tmp_len);
00324     if (w == 0U) { return 0U; }
00325     off += w;
00326     uint8_t tmp_s[32];
00327     tmp_len = trimLeadingZeros(tmp_s, sizeof(tmp_s), s, 32U);
00328     if (tmp_len == 0U) { return 0U; }
00329     w = RlpEncodeItem(buf + off, sizeof(buf) - off, tmp_s, (uint32_t)tmp_len);
00330     if (w == 0U) { return 0U; }
00331     off += w;
00332     size_t ret = rlpFinalize(out, outCap, buf, off);
00333     /* L-01: wipe the signature scratch buffers for hygiene uniformity with
00334      * the rest of the codebase. Signatures are public (broadcast on-chain)
00335      * so this is style/consistency only, not a real secret leak. */
00336     CW_Utils::secure_wipe(tmp_r, sizeof(tmp_r));
00337     CW_Utils::secure_wipe(tmp_s, sizeof(tmp_s));
00338     return ret;
00339 }
00340
00346 size_t encodeERC20Transfer(uint8_t* out) {
00347     out[0] = ERC20_TRANSFER_SEL_0; /* transfer(address,uint256) selector byte 0 */
00348     out[1] = ERC20_TRANSFER_SEL_1; /* transfer(address,uint256) selector byte 1 */
00349     out[2] = ERC20_TRANSFER_SEL_2; /* transfer(address,uint256) selector byte 2 */
00350     out[3] = ERC20_TRANSFER_SEL_3; /* transfer(address,uint256) selector byte 3 */
```

```
00351     CW_Utils::secure_wipe(out+4,  12U); /* bytes  4-15: ABI word padding before address (12 zero
      bytes) */
00352     if (!hexToBytes(ADDR_TO, out+16, 20)) {  /* bytes 16-35: recipient address (20 bytes)           */
00353         Serial.println(F("[fatal] ADDR_TO is not a valid 40-char hex string"));
00354         while (true) { /* halt to avoid broadcasting a wrong-recipient transfer */ }
00355     }
00356     CW_Utils::secure_wipe(out+36, 28U); /* bytes 36-63: ABI word padding before amount  (28 zero
      bytes) */
00357     out[ERC20_INDEX_OFFSET]   = (uint8_t)((AMOUNT_USDC » 24U) & 0xFFU);
00358     out[ERC20_INDEX_OFFSET+1] = (uint8_t)((AMOUNT_USDC » 16U) & 0xFFU);
00359     out[ERC20_INDEX_OFFSET+2] = (uint8_t)((AMOUNT_USDC » 8U)  & 0xFFU);
00360     out[ERC20_INDEX_OFFSET+3] = (uint8_t)( AMOUNT_USDC         & 0xFFU);
00361     return 68;
00362 }
00363
00370 bool sendRawTx(const uint8_t* raw, size_t len) {
00371     static const char requestPrefix[] =
00372         "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"eth_sendRawTransaction\","
00373         "\"params\":[\"0x";
00374     /* Fixed closing of the JSON-RPC body that follows the hex transaction bytes. */
00375     static const char requestSuffix[] = "\"]}";
00376     bool sent = false;
00377     for (uint8_t attempt = 0U; (attempt < TX_MAX_RETRIES) && !sent; attempt++) {
00378         if (attempt != 0U) {
00379             delay(TX_RETRY_DELAY_MS);
00380         }
00381         if (!ensureWiFi()) {
00382             Serial.println(F("sendRawTx: WiFi reconnect failed"));
00383             continue;
00384         }
00385         WiFiSSLClient wifiClient;
00386 #ifndef WIFI_DISABLE_CA_PINNING
00387         /* M-04: pin the RPC server's CA so a network MITM cannot serve a
00388          * forged certificate and feed crafted nonces/gas prices.
00389          * Define WIFI_DISABLE_CA_PINNING in config.h to skip this -- DEV ONLY,
00390          * leaves the connection vulnerable to MITM. */
00391         wifiClient.setCACert(WIFI_CA_CERT);
00392 #endif
00393         HttpClient client(wifiClient, RPC_HOST, RPC_PORT);
00394         client.beginRequest();
00395         int err = client.post(RPC_PATH);
00396         if (err != HTTP_SUCCESS) {
00397             Serial.print(F("sendRawTx: POST failed, err="));
00398             Serial.println(err);
00399             client.stop();
00400             continue;
00401         }
00402         /* Content-Length = prefix + 2 hex chars per raw byte + suffix. */
00403         client.sendHeader("Content-Type", "application/json");
00404         client.sendHeader("Content-Length",
00405             (int)(sizeof(requestPrefix)-1) + 2*(int)len + (int)(sizeof(requestSuffix)-1));
00406 #if defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET)
00407         {
00408             char authBuf[AUTH_HEADER_BUF_SIZE];
00409             buildBasicAuthHeader(authBuf, sizeof(authBuf));
00410             client.sendHeader("Authorization", authBuf);
00411         }
00412 #endif
00413         client.beginBody();
00414         client.print(requestPrefix);
00415         /* Encode each raw transaction byte as two hex characters and stream it
00416          * directly to the HTTP client, avoiding a large intermediate buffer. */
00417         char byteHexStr[HEX_CHAR_BUF_SIZE];
00418         byteHexStr[2] = '\0';
00419         for (size_t i = 0; i < len; i++) {
00420             byteHexStr[0] = hexChars[raw[i] » 4];   /* high nibble */
00421             byteHexStr[1] = hexChars[raw[i] & 0x0f]; /* low nibble  */
00422             client.print(byteHexStr);
00423         }
00424         client.print(requestSuffix);
00425         client.endRequest();
00426         int status = client.responseStatusCode();
00427         String responseBody = client.responseBody();
00428         Serial.print(F("[RPC] HTTP ")); Serial.println(status);
00429         /* NEW-2: dumping the full RPC response leaks tx metadata (nonce,
00430          * recipient, gas) over USB-CDC. Gate behind CW_DEBUG_LOGGING so a
00431          * production build (CW_DEBUG_LOGGING=0) stays silent. */
00432 #if CW_DEBUG_LOGGING
00433         Serial.print(F("[RPC] ")); Serial.println(responseBody);
00434 #endif
00435         bool statusOk = (status == HTTP_OK);
00436         /* L-03 (accepted): coarse substring search instead of a JSON parser.
00437          * Not a security issue -- at worst a false positive triggers a retry.
00438          * ArduinoJson would cost +15-25 KB flash for marginal robustness. */
00439         bool noJsonError = (responseBody.indexOf("\"error\"") == -1);
00440         sent = statusOk && noJsonError;
00441         /* eth_sendRawTransaction returns {"jsonrpc":"2.0","id":1,"result":"0x<txhash>"}.
```

```
00442          * The tx hash is on-chain public info -- extract and print so the user can
00443          * track the broadcast on a block explorer. */
00444         if (sent) {
00445             int r = responseBody.indexOf("\"result\":\"0x");
00446             if (r >= 0) {
00447                 int start = r + 10; /* skip past "\"result\":\"" */
00448                 int end = responseBody.indexOf("\"", start);
00449                 if (end > start) {
00450                     Serial.print(F("[tx] hash="));
00451                     Serial.println(responseBody.substring(start, end));
00452                 }
00453             }
00454         }
00455         client.stop();
00456     }
00457     return sent;
00458 }
00459
00471 uint8_t determineYParity(const uint8_t* hash, const uint8_t* r, const uint8_t* s) {
00472     /* ecrecover calldata: "0x" + hash(64) + v(64) + r(64) + s(64) = 258 chars + NUL */
00473     char hexBuf[260];
00474     uint16_t pos = 0U;
00475     hexBuf[pos++] = '0';
00476     hexBuf[pos++] = 'x';
00477     for (uint8_t i = 0U; i < 32U; i++) {
00478         hexBuf[pos++] = hexChars[hash[i] >> 4];
00479         hexBuf[pos++] = hexChars[hash[i] & 0x0f];
00480     }
00481     /* v field: ECRECOVER_V_PAD_CHARS zero chars, then 1 value byte -- filled per iteration */
00482     const uint16_t vOffset = pos;
00483     for (uint8_t i = 0U; i < ECRECOVER_V_PAD_CHARS; i++) {
00484         hexBuf[pos++] = '0';
00485     }
00486     pos += 2U; /* placeholder for v byte */
00487     for (uint8_t i = 0U; i < 32U; i++) {
00488         hexBuf[pos++] = hexChars[r[i] >> 4];
00489         hexBuf[pos++] = hexChars[r[i] & 0x0f];
00490     }
00491     for (uint8_t i = 0U; i < 32U; i++) {
00492         hexBuf[pos++] = hexChars[s[i] >> 4];
00493         hexBuf[pos++] = hexChars[s[i] & 0x0f];
00494     }
00495     hexBuf[pos] = '\0'; /* pos == 258 */
00496
00497     static const char requestPrefix[] =
00498         "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"eth_call\","
00499         "\"params\":[{\"to\":\"0x0000000000000000000000000000000000000001\","
00500         "\"data\":\"";
00501     static const char requestSuffix[] = "\"},\"latest\"]}";
00502     const int bodyLen = (int)(sizeof(requestPrefix) - 1) + 258 + (int)(sizeof(requestSuffix) - 1);
00503
00504     uint8_t result = YPARITY_UNKNOWN;
00505     for (uint8_t yp = 0U; (yp <= 1U) && (result == YPARITY_UNKNOWN); yp++) {
00506         /* Patch v byte into last two chars of the v field.
00507          * Ethereum ecrecover: v=27 means yParity=0, v=28 means yParity=1. */
00508         const uint8_t v = ECRECOVER_V_BASE + yp;
00509         hexBuf[vOffset + ECRECOVER_V_PAD_CHARS]     = hexChars[(v & 0xFFU) >> 4U];
00510         hexBuf[vOffset + ECRECOVER_V_PAD_CHARS + 1U] = hexChars[(v & 0xFFU) & 0x0FU];
00511
00512         if (!ensureWiFi()) {
00513             Serial.println(F("determineYParity: WiFi reconnect failed"));
00514             continue;
00515         }
00516         WiFiSSLClient wifiClient;
00517 #ifndef WIFI_DISABLE_CA_PINNING
00518         /* M-04: pin the RPC server's CA so a network MITM cannot serve a
00519          * forged certificate and feed crafted nonces/gas prices.
00520          * Define WIFI_DISABLE_CA_PINNING in config.h to skip this -- DEV ONLY,
00521          * leaves the connection vulnerable to MITM. */
00522         wifiClient.setCACert(WIFI_CA_CERT);
00523 #endif
00524         HttpClient client(wifiClient, RPC_HOST, RPC_PORT);
00525         client.beginRequest();
00526         int err = client.post(RPC_PATH);
00527         if (err != HTTP_SUCCESS) {
00528             Serial.print(F("determineYParity: POST failed, err="));
00529             Serial.println(err);
00530             client.stop();
00531             continue;
00532         }
00533         client.sendHeader("Content-Type", "application/json");
00534         client.sendHeader("Content-Length", bodyLen);
00535 #if defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET)
00536         {
00537             char authBuf[AUTH_HEADER_BUF_SIZE];
00538             buildBasicAuthHeader(authBuf, sizeof(authBuf));
00539             client.sendHeader("Authorization", authBuf);
```

```
00540         }
00541 #endif
00542         client.beginBody();
00543         client.print(requestPrefix);
00544         client.print(hexBuf);
00545         client.print(requestSuffix);
00546         client.endRequest();
00547
00548         int status = client.responseStatusCode();
00549         String response = client.responseBody(); /* consume response body */
00550         client.stop();
00551         if (status != HTTP_OK) {
00552             continue;
00553         }
00554         int resultIdx = response.indexOf("\"result\"");
00555         if (resultIdx < 0) {
00556             continue;
00557         }
00558         int hexIdx = response.indexOf("0x", resultIdx);
00559         if (hexIdx < 0) {
00560             continue;
00561         }
00562         /* H-05: validate the response is long enough before substring(). ecrecover
00563          * returns a 32-byte (64-hex-char) word; with the "0x" prefix the slice
00564          * spans hexIdx+2 .. hexIdx+66. A truncated RPC response would silently
00565          * give us an empty/garbage recovered address and let the loop progress. */
00566         if (response.length() < (unsigned int)(hexIdx + 66)) {
00567             continue;
00568         }
00569         /* ecrecover returns 32-byte word; address = last 20 bytes = last 40 hex chars */
00570         String recovered = response.substring(hexIdx + 26, hexIdx + 66);
00571         Serial.print(F("[ecrecover] v=")); Serial.print(v);
00572         Serial.print(F(" recovered=0x")); Serial.println(recovered);
00573         Serial.print(F("[ecrecover] expected=0x")); Serial.println(F(ADDR_FROM));
00574         if (recovered.equalsIgnoreCase(ADDR_FROM)) {
00575             result = yp;
00576         }
00577     }
00578     return result;
00579 }
00580
00586 uint8_t fetchNonce(uint64_t* nonce) {
00587     static const char requestPrefix[] =
00588         "{\"jsonrpc\":\"2.0\",\"id\":2,\"method\":\"eth_getTransactionCount\","
00589         "\"params\":[\"0x";
00590     static const char requestSuffix[] = "\",\"pending\"]}";
00591     const int bodyLen = (int)(sizeof(requestPrefix)-1) + 40 + (int)(sizeof(requestSuffix)-1);
00592
00593     uint8_t result = 1U;
00594     for (uint8_t attempt = 0U; (attempt < TX_MAX_RETRIES) && (result != 0U); attempt++) {
00595         if (attempt != 0U) {
00596             delay(1000U);
00597         }
00598         if (!ensureWiFi()) {
00599             Serial.println(F("fetchNonce: WiFi reconnect failed"));
00600             continue;
00601         }
00602         WiFiSSLClient wifiClient;
00603 #ifndef WIFI_DISABLE_CA_PINNING
00604         /* M-04: pin the RPC server's CA so a network MITM cannot serve a
00605          * forged certificate and feed crafted nonces/gas prices.
00606          * Define WIFI_DISABLE_CA_PINNING in config.h to skip this -- DEV ONLY,
00607          * leaves the connection vulnerable to MITM. */
00608         wifiClient.setCACert(WIFI_CA_CERT);
00609 #endif
00610         HttpClient client(wifiClient, RPC_HOST, RPC_PORT);
00611         client.beginRequest();
00612         Serial.print(F("fetchNonce: connecting to "));
00613         Serial.print(F(RPC_HOST));
00614         Serial.println(F(RPC_PATH));
00615         int err = client.post(RPC_PATH);
00616         if (err != HTTP_SUCCESS) {
00617             Serial.print(F("fetchNonce: POST failed, err="));
00618             Serial.println(err);
00619             client.stop();
00620             continue;
00621         }
00622         client.sendHeader("Content-Type", "application/json");
00623         client.sendHeader("Content-Length", bodyLen);
00624 #if defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET)
00625         {
00626             char authBuf[AUTH_HEADER_BUF_SIZE];
00627             buildBasicAuthHeader(authBuf, sizeof(authBuf));
00628             client.sendHeader("Authorization", authBuf);
00629         }
00630 #endif
00631         client.beginBody();
```

```
00632         client.print(requestPrefix);
00633         client.print(ADDR_FROM);
00634         client.print(requestSuffix);
00635         client.endRequest();
00636
00637         int status = client.responseStatusCode();
00638         String resp = client.responseBody();
00639         client.stop();
00640         if (status == HTTP_OK) {
00641             int ri = resp.indexOf("\"result\"");
00642             int xi = (ri >= 0) ? resp.indexOf("0x", ri) : -1;
00643             if (xi >= 0) {
00644                 uint64_t parsed = 0U;
00645                 /* H-06: cap at 16 hex digits (uint64 capacity). A malicious or
00646                  * malformed RPC returning a longer hex string would silently
00647                  * overflow the uint64 and wrap to a small (replayed) nonce. */
00648                 int digitCount = 0;
00649                 bool overflowed = false;
00650                 for (int i = xi + 2; i < (int)resp.length(); i++) {
00651                     char c = resp[i];
00652                     if (!((c>='0'&&c<='9')||(c>='a'&&c<='f')||(c>='A'&&c<='F'))) break;
00653                     if (digitCount >= 16) { overflowed = true; break; }
00654                     parsed = (parsed « 4) | fromHex(c);
00655                     digitCount++;
00656                 }
00657                 /* NEW-4: require >= 1 hex digit so "0x" / "0xZZZ" is not
00658                  * treated as a valid nonce=0 result on parse failure. */
00659                 if (!overflowed && (digitCount > 0)) {
00660                     *nonce = parsed;
00661                     result = 0U;
00662                 }
00663             }
00664         }
00665     }
00666     return result;
00667 }
00668
00672 void setup() {
00673     Serial.begin(115200);
00674     delay(2000);
00675
00676     /* Init SPI and PN532 */
00677     SPI.begin();
00678     if (!wallet.begin()) {
00679         Serial.println(F("PN532 init failed! Halting."));
00680         while(1);
00681     }
00682     Serial.println(F("PN532 OK"));
00683
00684 #ifdef WIFI_DISABLE_CA_PINNING
00685     Serial.println(F("  WIFI_DISABLE_CA_PINNING is set -- TLS certificate is NOT validated."));
00686     Serial.println(F("    Connection is vulnerable to MITM. DEV ONLY, do not use in production."));
00687 #endif
00688
00689     /* Connect to WiFi */
00690     Serial.print(F("Connecting to WiFi"));
00691     WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
00692     uint8_t retries = 20U;
00693     while ((retries > 0U) && (WiFi.status() != WL_CONNECTED)) {
00694         retries--;
00695         delay(500U);
00696         Serial.print(F("."));
00697     }
00698     Serial.println();
00699     if (WiFi.status() != WL_CONNECTED) {
00700         Serial.println(F("WiFi failed!"));
00701         while(1);
00702     }
00703     delay(2000); /* Allow network stack to stabilise before first SSL connection */
00704
00705     /* Build ERC-20 calldata */
00706     uint8_t calldata[68];
00707     size_t calLen = encodeERC20Transfer(calldata);
00708
00709     /* Build unsigned EIP-1559 transaction */
00710     Tx2 tx2;
00711     uint64_t fetchedNonce = 0U;
00712     if (fetchNonce(&fetchedNonce) != 0U) {
00713         Serial.println(F("fetchNonce failed! Halting."));
00714         while(1);
00715     }
00716     tx2.nonce             = fetchedNonce;
00717     tx2.maxPriorityFeePerGas = MAX_PRIORITY_FEE;
00718     tx2.maxFeePerGas      = MAX_FEE;
00719     tx2.gasLimit          = GAS_LIMIT_ERC20;
00720     tx2.to                = ADDR_USDC;
00721     tx2.value             = 0;
```

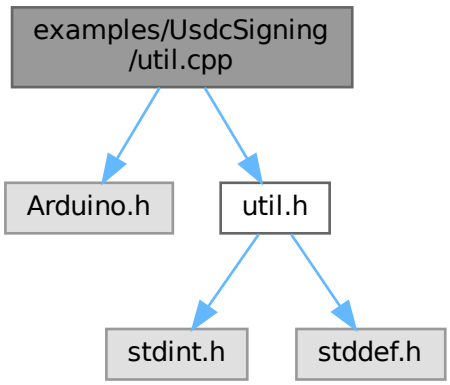
```
00722     tx2.data               = calldata;
00723     tx2.dataLen            = calLen;
00724     tx2.chainId            = CHAIN_ID_SEPOLIA;
00725
00726     /* RLP encode unsigned tx */
00727     static const size_t kRlpBufSize = 512U;
00728     uint8_t rlpUnsigned[kRlpBufSize];
00729     size_t rlpLen = rlpEncodeUnsignedTx(tx2, rlpUnsigned, kRlpBufSize);
00730     /* L-02: defense-in-depth. A USDC transfer EIP-1559 tx is ~150-200 bytes,
00731      * so 512 is plenty for the current shape -- but if a future change adds
00732      * an access list, auth list (EIP-7702), or larger tx.data, a silent
00733      * stack overflow would corrupt the return address. Halt explicitly. */
00734     /* rlpFinalize returns 0 on overflow (via safe_memcpy bounds check). The
00735      * post-check stays as defense-in-depth in case a future change skips
00736      * rlpFinalize's gate. */
00737     if ((rlpLen == 0U) || (rlpLen > kRlpBufSize)) {
00738         Serial.print(F("[fatal] rlpUnsigned overflow or empty: ")); Serial.println(rlpLen);
00739         while (true) {}
00740     }
00741
00742     /* Keccak-256 hash */
00743     uint8_t hashKeccak[32];
00744     keccak256(static_cast<const uint8_t*>(rlpUnsigned), rlpLen, hashKeccak);
00745     printHex("[HASH]", hashKeccak, 32);
00746
00747     /* BIP44 Ethereum path: m/44'/60'/0'/0/0 */
00748     static const uint8_t eth_path[20] = {
00749         0x80U,0x00U,0x00U,0x2CU,  /* 44' */
00750         0x80U,0x00U,0x00U,0x3CU,  /* 60' */
00751         0x80U,0x00U,0x00U,0x00U,  /* 0'  */
00752         0x00U,0x00U,0x00U,0x00U,  /* 0   */
00753         0x00U,0x00U,0x00U,0x00U,  /* 0   */
00754     };
00755
00756     /* === Sign with Cryptnox card over NFC (with retry on NFC dropout) === */
00757     Serial.println(F("Place Cryptnox card on PN532 reader..."));
00758     CW_SignResult signResult;
00759     for (uint8_t attempt = 0U; attempt < 3U; attempt++) {
00760         if (attempt != 0U) {
00761             delay(1000U);
00762         }
00763         CW_SecureSession session;
00764         if (!wallet.connect(session)) {
00765             continue;
00766         }
00767         Serial.println(F("Card connected, secure channel established."));
00768
00769         CW_SignRequest signReq(session, CW_SIGN_DERIVE_K1, CW_SIGN_SIG_ECDSA_LOW_S, CW_SIGN_WITH_PIN);
00770         signReq.hash             = hashKeccak;
00771         signReq.hashLength       = static_cast<uint8_t>(CW_HASH_SIZE);
00772         signReq.derivePath       = eth_path;
00773         signReq.derivePathLength = static_cast<uint8_t>(sizeof(eth_path));
00774         (void)CW_Utils::safe_memcpy(signReq.pin, sizeof(signReq.pin),
00775                                     reinterpret_cast<const uint8_t*>(CARD_PIN), CARD_PIN_LEN);
00776
00777         signResult = wallet.sign(signReq);
00778         wallet.disconnect(session);
00779         if (signResult.errorCode == CW_OK) {
00780             break;
00781         }
00782         if (signResult.errorCode == CW_SIGN_PIN_INCORRECT ||
00783             signResult.errorCode == CW_SIGN_NO_KEY_LOADED) {
00784             Serial.print(F("Card rejected sign command (error=0x"));
00785             Serial.print(signResult.errorCode, HEX);
00786             Serial.println(F(") -- check PIN and card initialisation. Halting."));
00787             /* M-05: explicitly wipe the PIN before halting. The CW_SignRequest
00788              * destructor would normally do this on scope exit, but the while(1)
00789              * below stays inside the for() body so the destructor never runs. */
00790             CW_Utils::secure_wipe(signReq.pin, sizeof(signReq.pin));
00791             while(1);
00792         }
00793         Serial.print(F("Sign attempt "));
00794         Serial.print(attempt + 1U);
00795         Serial.print(F(" failed, error=0x"));
00796         Serial.println(signResult.errorCode, HEX);
00797     }
00798
00799     if (signResult.errorCode != CW_OK) {
00800         Serial.print(F("Sign failed: error=0x"));
00801         Serial.println(signResult.errorCode, HEX);
00802         while(1);
00803     }
00804     Serial.println(F("Signed."));
00805
00806     const uint8_t* r = signResult.signature;       /* first 32 bytes */
00807     const uint8_t* s = signResult.signature + 32;  /* last  32 bytes */
00808     printHex("[SIG r]", r, 32);
```

```
00809     printHex("[SIG s]", s, 32);
00810
00811     /* Determine yParity */
00812     uint8_t yParity = determineYParity(hashKeccak, r, s);
00813     if (yParity == YPARITY_UNKNOWN) {
00814         Serial.println(F("yParity determination failed! Halting."));
00815         while(1);
00816     }
00817     Serial.print(F("yParity: "));
00818     Serial.println(yParity);
00819
00820     /* RLP encode signed tx and send */
00821     uint8_t rlpSigned[kRlpBufSize];
00822     size_t rlpSignedLen = rlpEncodeSignedTx(tx2, r, s, &yParity, rlpSigned, kRlpBufSize);
00823     /* L-02 + safe_memcpy gate inside rlpFinalize. */
00824     if ((rlpSignedLen == 0U) || (rlpSignedLen > kRlpBufSize)) {
00825         Serial.print(F("[fatal] rlpSigned overflow or empty: ")); Serial.println(rlpSignedLen);
00826         while (true) {}
00827     }
00828
00829     Serial.println(F("Sending..."));
00830     if (sendRawTx(rlpSigned, rlpSignedLen)) {
00831         Serial.println(F("Transaction sent successfully!"));
00832     } else {
00833         Serial.println(F("Transaction FAILED."));
00834     }
00835 }
00836
00840 void loop() {}
```

6.15 examples/UsdcSigning/util.cpp File Reference

```
#include <Arduino.h>
#include "util.h"
```

Include dependency graph for util.cpp:



Functions

- int fromHex (char c)

  Convert a hexadecimal character to a byte value.
- bool hexToBytes (const char *hex, uint8_t *out, size_t len)

  Convert a hex string to a byte array.
- uint32_t RlpEncodeWholeHeader (uint8_t *header_output, size_t header_cap, uint32_t total_len)

  Encodes the RLP list header for a sequence of items.
- uint32_t RlpEncodeItem (uint8_t *output, size_t output_cap, const uint8_t *input, uint32_t input↩ _len)

Encodes a single RLP item.

- uint32_t ConvertNumberToUintArray (uint8_t ∗str, uint64_t val)

  Converts an unsigned integer into a big-endian byte array.

- size_t trimLeadingZeros (uint8_t ∗out, size_t out_cap, const uint8_t ∗in, size_t in_len)

  Trims leading zeros from a byte array.

6.15.1 Function Documentation

6.15.1.1 ConvertNumberToUintArray()

```
uint32_t ConvertNumberToUintArray (
            uint8_t * str,
            uint64_t val)
```

Converts an unsigned integer into a big-endian byte array.
Converts an unsigned integer to a big-endian byte array.
NEW-5: widened to uint64_t. Previously truncated Tx2 uint64 fields (nonce/fees/gasLimit/value) at 4 bytes, producing wrong-amount tx.

Parameters

out	*str*	Output buffer (at least 8 bytes).
in	*val*	Unsigned 64-bit integer to convert.

Returns

Number of bytes written to the buffer (1..8).

Examples

UsdcSigning.ino.

Definition at line 190 of file util.cpp.
Referenced by rlpEncodeTxBody().

6.15.1.2 fromHex()

```
int fromHex (
            char c)
```

Convert a hexadecimal character to a byte value.
M-07: returns -1 on any non-hex character so callers can distinguish a true '0' from a parse error.

Examples

UsdcSigning.ino.

Definition at line 15 of file util.cpp.
Referenced by fetchNonce(), and hexToBytes().

6.15.1.3 hexToBytes()

```
bool hexToBytes (
            const char * hex,
            uint8_t * out,
            size_t len)
```

Convert a hex string to a byte array.
H-03: validate input length before any read. Without this guard a truncated or malicious RPC response shorter than 2∗len would cause out-of-bounds reads past the NUL terminator.

Returns

true on success, false on NULL input or insufficient length.

Examples

UsdcSigning.ino.

Definition at line 32 of file util.cpp.
References fromHex().
Referenced by encodeERC20Transfer(), and rlpEncodeTxBody().

6.15.1.4 RlpEncodeItem()

```
uint32_t RlpEncodeItem (
            uint8_t * output,
            size_t output_cap,
            const uint8_t * input,
            uint32_t input_len)
```

Encodes a single RLP item.
Encodes a single item in RLP format.

Parameters

out	*output*	Buffer where the encoded RLP item will be written.
in	*input*	Input data to encode.
in	*input_len*	Length of input data.

Returns

Number of bytes written to output.

Examples

UsdcSigning.ino.

Definition at line 123 of file util.cpp.
Referenced by rlpEncodeSignedTx().

6.15.1.5 RlpEncodeWholeHeader()

```
uint32_t RlpEncodeWholeHeader (
            uint8_t * header_output,
            size_t header_cap,
            uint32_t total_len)
```

Encodes the RLP list header for a sequence of items.
Encodes the total length into an RLP list header.
This function generates the RLP "whole header" for a list payload of length total_len, according to Ethereum RLP specification.
RLP encoding rules:

- For a list with total payload < 55 bytes: 0xC0 + total_len (single-byte header)
- For a list with payload >= 55 bytes: 0xF7 + length_of_length_field, followed by the big-endian encoded total_len

Example: total_len = 10 → header = C0 + 0x0A → C0 0A total_len = 100 → total_len = 0x64 (1 byte) → F8 64 total_len = 1024 → total_len = 0x0400 (2 bytes) → F9 04 00
This header is intended to be placed immediately before the concatenated items in an RLP list. For EIP-1559 typed transactions, it must appear after the type byte (0x02).

Parameters

out	*header_output*	Pointer where the encoded header will be written.
in	*total_len*	Length in bytes of all list items concatenated.

Returns

Number of bytes written to header_output.

Examples

UsdcSigning.ino.

Definition at line 76 of file util.cpp.
Referenced by rlpFinalize().

6.15.1.6 trimLeadingZeros()

```
size_t trimLeadingZeros (
            uint8_t * out,
            size_t out_cap,
            const uint8_t * in,
            size_t in_len)
```

Trims leading zeros from a byte array.
Removes leading zeros from a byte array.

Parameters

out	*out*	Output buffer.
in	*in*	Input buffer.
in	*in_len*	Length of input buffer.

Returns

Number of bytes written to the output buffer.

Examples

UsdcSigning.ino.

Definition at line 220 of file util.cpp.
Referenced by rlpEncodeSignedTx().

6.16 util.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 <Arduino.h>
00007 #include "util.h"
00008
00015 int fromHex(char c) {
00016     if (c >= '0' && c <= '9') return c - '0';
00017     if (c >= 'a' && c <= 'f') return c - 'a' + 10;
00018     if (c >= 'A' && c <= 'F') return c - 'A' + 10;
00019     return -1;
00020 }
00021
00031 // cppcheck-suppress unusedFunction
```

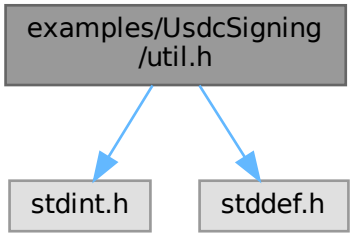
```
00032 bool hexToBytes(const char* hex, uint8_t* out, size_t len) {
00033     if ((hex == nullptr) || (out == nullptr)) {
00034         return false;
00035     }
00036     if (strlen(hex) < (2U * len)) {
00037         return false;
00038     }
00039     for (size_t i = 0; i < len; i++) {
00040         int hi = fromHex(hex[2*i]);
00041         int lo = fromHex(hex[2*i+1]);
00042         /* M-07: reject invalid characters instead of silently mapping them to 0. */
00043         if ((hi < 0) || (lo < 0)) {
00044             return false;
00045         }
00046         out[i] = (uint8_t)((hi « 4) | lo);
00047     }
00048     return true;
00049 }
00050
00075 // cppcheck-suppress unusedFunction
00076 uint32_t RlpEncodeWholeHeader(uint8_t* header_output, size_t header_cap, uint32_t total_len)
00077 {
00078     if (total_len < 55U)
00079     {
00080         if (header_cap < 1U) { return 0U; }
00081         header_output[0] = (uint8_t)0xc0 + (uint8_t)total_len;
00082         return 1U;
00083     }
00084     else
00085     {
00086         uint8_t tmp_header[8];
00087         memset(tmp_header, 0, 8U);
00088         uint32_t hexdigit = 1U;
00089         uint32_t tmp = total_len;
00090         while ((uint32_t)(tmp / 256U) > 0U)
00091         {
00092             tmp_header[hexdigit] = (uint8_t)(tmp % 256U);
00093             tmp = (uint32_t)(tmp / 256U);
00094             hexdigit++;
00095         }
00096         tmp_header[hexdigit] = (uint8_t)(tmp);
00097         tmp_header[0] = (uint8_t)0xf7 + (uint8_t)hexdigit;
00098
00099         // fix direction for header
00100         uint8_t header[8];
00101         memset(header, 0, 8U);
00102         header[0] = tmp_header[0];
00103         for (uint32_t i = 0U; i < hexdigit; i++)
00104         {
00105             header[i + 1U] = tmp_header[hexdigit - i];
00106         }
00107         const size_t need = (size_t)hexdigit + 1U;
00108         if (need > header_cap) { return 0U; }
00109         memcpy(header_output, header, need);
00110         return hexdigit + 1U;
00111     }
00112 }
00113
00122 // cppcheck-suppress unusedFunction
00123 uint32_t RlpEncodeItem(uint8_t* output, size_t output_cap, const uint8_t* input, uint32_t input_len)
00124 {
00125     if (input_len == 1U && input[0] == 0x00U)
00126     {
00127         if (output_cap < 1U) { return 0U; }
00128         const uint8_t c[1] = {0x80};
00129         memcpy(output, c, 1U);
00130         return 1U;
00131     }
00132     else if (input_len == 1U && input[0] < 128U)
00133     {
00134         if (output_cap < 1U) { return 0U; }
00135         memcpy(output, input, 1U);
00136         return 1U;
00137     }
00138     else if (input_len <= 55U)
00139     {
00140         const size_t need = (size_t)input_len + 1U;
00141         if (need > output_cap) { return 0U; }
00142         const uint8_t _ = (uint8_t)0x80 + (uint8_t)input_len;
00143         const uint8_t header[] = {_};
00144         memcpy(output, header, 1U);
00145         memcpy(output + 1U, input, (size_t)input_len);
00146         return input_len + 1U;
00147     }
00148     else
00149     {
00150         uint8_t tmp_header[8];
```

```
00151         memset(tmp_header, 0, 8U);
00152         uint32_t hexdigit = 1U;
00153         uint32_t tmp = input_len;
00154         while ((uint32_t)(tmp / 256U) > 0U)
00155         {
00156             tmp_header[hexdigit] = (uint8_t)(tmp % 256U);
00157             tmp = (uint32_t)(tmp / 256U);
00158             hexdigit++;
00159         }
00160         tmp_header[hexdigit] = (uint8_t)(tmp);
00161         tmp_header[0] = (uint8_t)0xb7 + (uint8_t)hexdigit;
00162 
00163         // fix direction for header
00164         uint8_t header[8];
00165         memset(header, 0, 8U);
00166         header[0] = tmp_header[0];
00167         for (uint32_t i = 0U; i < hexdigit; i++)
00168         {
00169             header[i + 1U] = tmp_header[hexdigit - i];
00170         }
00171         const size_t need = (size_t)input_len + hexdigit + 1U;
00172         if (need > output_cap) { return 0U; }
00173         memcpy(output, header, hexdigit + 1U);
00174         memcpy(output + hexdigit + 1U, input, (size_t)input_len);
00175         return input_len + hexdigit + 1U;
00176     }
00177 }
00178 
00189 // cppcheck-suppress unusedFunction
00190 uint32_t ConvertNumberToUintArray(uint8_t* str, uint64_t val)
00191 {
00192     uint32_t ret = 0U;
00193     uint8_t tmp[8];
00194     memset(tmp, 0, 8U);
00195 
00196     while ((val / 256ULL) > 0ULL)
00197     {
00198         tmp[ret] = (uint8_t)(val % 256ULL);
00199         val = val / 256ULL;
00200         ret++;
00201     }
00202     tmp[ret] = (uint8_t)(val % 256ULL);
00203     for (uint32_t i = 0U; i < ret + 1U; i++)
00204     {
00205         str[i] = tmp[ret - i];
00206     }
00207 
00208     return ret + 1U;
00209 }
00210 
00219 // cppcheck-suppress unusedFunction
00220 size_t trimLeadingZeros(uint8_t* out, size_t out_cap, const uint8_t* in, size_t in_len)
00221 {
00222     if ((in_len == 0U) || (out_cap == 0U)) { return 0U; }
00223     size_t start = 0;
00224     while (start < in_len - 1 && in[start] == 0)
00225     {
00226         start++;
00227     }
00228     size_t len = in_len - start;
00229     /* cppcheck-suppress knownConditionTrueFalse
00230      * cppcheck only follows the loop-exit path where start == in_len-1
00231      * (giving len == 1) and misses the early exit when in[start] != 0
00232      * (giving len up to in_len). The check is genuine defense-in-depth
00233      * if a future caller passes out_cap < in_len. */
00234     if (len > out_cap) { return 0U; }
00235     memcpy(out, in + start, len);
00236     return len;
00237 }
```

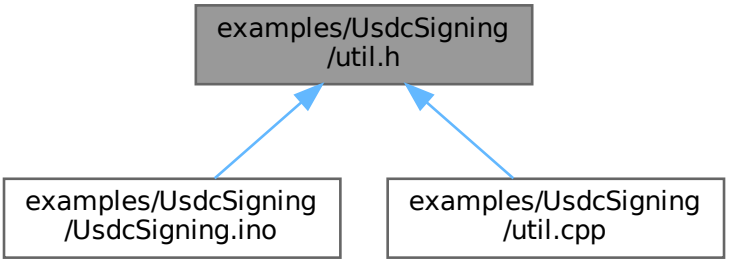
6.17 examples/UsdcSigning/util.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for util.h:



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



Functions

- uint32_t RlpEncodeWholeHeader (uint8_t *header_output, size_t header_cap, uint32_t total_len)

  Encodes the total length into an RLP list header.
- uint32_t RlpEncodeItem (uint8_t *output, size_t output_cap, const uint8_t *input, uint32_t input↩ _len)

  Encodes a single item in RLP format.
- uint32_t ConvertNumberToUintArray (uint8_t *str, uint64_t val)

  Converts an unsigned integer to a big-endian byte array.
- size_t trimLeadingZeros (uint8_t *out, size_t out_cap, const uint8_t *in, size_t in_len)

  Removes leading zeros from a byte array.
- int fromHex (char c)

  Convert a hexadecimal character to a byte value.
- bool hexToBytes (const char *hex, uint8_t *out, size_t len)

  Convert a hex string to a byte array.

6.17.1 Function Documentation

6.17.1.1 ConvertNumberToUintArray()

```
uint32_t ConvertNumberToUintArray (
            uint8_t * str,
            uint64_t val)
```

Converts an unsigned integer to a big-endian byte array.
NEW-5: widened from uint32_t to uint64_t so that 64-bit Ethereum fields (nonce, gasLimit, value, maxFeePerGas, maxPriorityFeePerGas) are not silently truncated. Returns up to 8 bytes; `str` must be >= 8 bytes.

Parameters

`out`	*str*	Pointer to output buffer (at least 8 bytes).
`in`	*val*	Unsigned 64-bit integer to convert.

Returns

Number of bytes used in the output array (1..8).

Converts an unsigned integer to a big-endian byte array.
NEW-5: widened to uint64_t. Previously truncated Tx2 uint64 fields (nonce/fees/gasLimit/value) at 4 bytes, producing wrong-amount tx.

Parameters

`out`	*str*	Output buffer (at least 8 bytes).
`in`	*val*	Unsigned 64-bit integer to convert.

Returns

Number of bytes written to the buffer (1..8).

Definition at line 190 of file util.cpp.
Referenced by rlpEncodeTxBody().

6.17.1.2 fromHex()

```
int fromHex (
            char c)
```

Convert a hexadecimal character to a byte value.
Returns -1 on any non-hex character so callers can distinguish a true '0' from a parse error. Previously this returned 0 on invalid input, causing silent data corruption (security audit M-07).

Parameters

c	Hex character.

Returns

0..15 for valid hex digits, -1 otherwise.

M-07: returns -1 on any non-hex character so callers can distinguish a true '0' from a parse error.
Definition at line 15 of file util.cpp.
Referenced by fetchNonce(), and hexToBytes().

6.17.1.3 hexToBytes()

```
bool hexToBytes (
            const char * hex,
            uint8_t * out,
            size_t len)
```

Convert a hex string to a byte array.
Validates that `hex` contains at least `2*len` characters before any read, to prevent out-of-bounds reads on malformed / truncated RPC responses (security audit H-03).

Parameters

hex	Input hex string (must be NUL-terminated).
out	Output byte array (caller-allocated, at least `len` bytes).
len	Number of bytes to write to `out`.

Returns

true on success, false if `hex` is too short or NULL.

H-03: validate input length before any read. Without this guard a truncated or malicious RPC response shorter than 2∗len would cause out-of-bounds reads past the NUL terminator.

Returns

true on success, false on NULL input or insufficient length.

Definition at line 32 of file util.cpp.
References fromHex().
Referenced by encodeERC20Transfer(), and rlpEncodeTxBody().

6.17.1.4 RlpEncodeItem()

```
uint32_t RlpEncodeItem (
            uint8_t * output,
            size_t output_cap,
            const uint8_t * input,
            uint32_t input_len)
```

Encodes a single item in RLP format.
Bounds-checked: returns 0 (and does not write) if the encoded item would not fit in `output_cap` bytes.

Parameters

`out`	*output*	Pointer to buffer to store the RLP-encoded item.
`in`	*output_cap*	Capacity of `output` in bytes.
`in`	*input*	Pointer to the data to encode.
`in`	*input_len*	Length of the input data in bytes.

Returns

Number of bytes written into output, or 0 on overflow.

Encodes a single item in RLP format.

Parameters

out	*output*	Buffer where the encoded RLP item will be written.
in	*input*	Input data to encode.
in	*input_len*	Length of input data.

Returns

Number of bytes written to output.

Definition at line 123 of file util.cpp.
Referenced by rlpEncodeSignedTx().

6.17.1.5 RlpEncodeWholeHeader()

```
uint32_t RlpEncodeWholeHeader (
            uint8_t * header_output,
            size_t header_cap,
            uint32_t total_len)
```

Encodes the total length into an RLP list header.
Bounds-checked: returns 0 (and does not write) if the encoded header would not fit in `header_cap` bytes.

Parameters

out	*header_output*	Pointer to buffer to store the RLP header.
in	*header_cap*	Capacity of `header_output` in bytes.
in	*total_len*	Total length of the RLP list content.

Returns

Number of bytes written into header_output, or 0 on overflow.

Encodes the total length into an RLP list header.
This function generates the RLP "whole header" for a list payload of length total_len, according to Ethereum RLP specification.
RLP encoding rules:

- For a list with total payload $<$ 55 bytes: 0xC0 + total_len (single-byte header)
- For a list with payload $>=$ 55 bytes: 0xF7 + length_of_length_field, followed by the big-endian encoded total_len

Example: total_len = 10 → header = C0 + 0x0A → C0 0A total_len = 100 → total_len = 0x64 (1 byte) → F8 64 total_len = 1024 → total_len = 0x0400 (2 bytes) → F9 04 00
This header is intended to be placed immediately before the concatenated items in an RLP list. For EIP-1559 typed transactions, it must appear after the type byte (0x02).

Parameters

out	*header_output*	Pointer where the encoded header will be written.
in	*total_len*	Length in bytes of all list items concatenated.

Returns

Number of bytes written to header_output.

Definition at line 76 of file util.cpp.
Referenced by rlpFinalize().

6.17.1.6 trimLeadingZeros()

```
size_t trimLeadingZeros (
            uint8_t * out,
            size_t out_cap,
            const uint8_t * in,
            size_t in_len)
```

Removes leading zeros from a byte array.
Bounds-checked: returns 0 if `out_cap` is smaller than the trimmed length. (Trimmed length is always `<= in_len`.)

Parameters

`out`	*out*	Pointer to buffer to store trimmed result.
`in`	*out_cap*	Capacity of `out` in bytes.
`in`	*in*	Pointer to input byte array.
`in`	*in_len*	Length of input array.

Returns

Number of bytes written into out, or 0 on overflow.

Removes leading zeros from a byte array.

Parameters

`out`	*out*	Output buffer.
`in`	*in*	Input buffer.
`in`	*in_len*	Length of input buffer.

Returns

Number of bytes written to the output buffer.

Definition at line 220 of file util.cpp.
Referenced by rlpEncodeSignedTx().

6.18 util.h

Go to the documentation of this file.

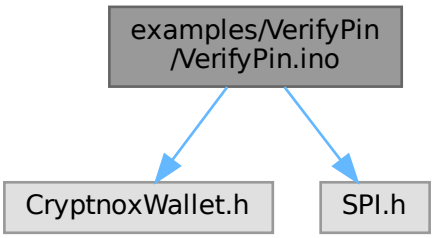
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00006 #ifndef UTIL_H
00007 #define UTIL_H
00008
00009 #include <stdint.h>
00010 #include <stddef.h>
00011
00023 uint32_t RlpEncodeWholeHeader(uint8_t *header_output, size_t header_cap, uint32_t total_len);
```

```
00024
00037 uint32_t RlpEncodeItem(uint8_t* output, size_t output_cap, const uint8_t* input, uint32_t input_len);
00038
00050 uint32_t ConvertNumberToUintArray(uint8_t *str, uint64_t val);
00051
00064 size_t trimLeadingZeros(uint8_t* out, size_t out_cap, const uint8_t* in, size_t in_len);
00065
00076 int fromHex(char c);
00077
00090 bool hexToBytes(const char* hex, uint8_t* out, size_t len);
00091
00092 #endif /* UTIL_H */
```

6.19 examples/VerifyPin/VerifyPin.ino File Reference

```
#include <CryptnoxWallet.h>
#include <SPI.h>
```

Include dependency graph for VerifyPin.ino:



Macros

- #define PN532_SS_PIN (10U)

  SPI slave-select (CS) pin connected to the PN532 module.
- #define DEMO_PIN "000000000"

  Demo PIN used by this example. Must match the PIN that the card was initialised with (4–9 ASCII digits).

Functions

- PN532Adapter nfc (serialAdapter, PN532_SS_PIN, &SPI)

  PN532 transport adapter over SPI.
- void setup ()

  Arduino setup hook.
- void loop ()

  Arduino main loop.

Variables

- ArduinoLoggerAdapter serialAdapter

  Arduino logger adapter — emits SDK diagnostics on `Serial`.
- ArduinoCryptoProvider cryptoProvider

  Crypto provider (AES / SHA / micro-ecc / TRNG bridge for Arduino).
- ArduinoPlatform platform

  Platform adapter (Arduino blocking delay).

- CryptnoxWallet wallet (nfc, serialAdapter, cryptoProvider, platform)

  High-level Cryptnox wallet wiring the four adapters together.

6.19.1 Macro Definition Documentation

6.19.1.1 DEMO_PIN

```
#define DEMO_PIN "000000000"
```

Demo PIN used by this example. Must match the PIN that the card was initialised with (4–9 ASCII digits).
Definition at line 47 of file VerifyPin.ino.

6.19.1.2 PN532_SS_PIN

```
#define PN532_SS_PIN (10U)
```

SPI slave-select (CS) pin connected to the PN532 module.
Definition at line 41 of file VerifyPin.ino.

6.19.2 Function Documentation

6.19.2.1 loop()

```
void loop ()
```

Arduino main loop.
One full secure-channel session per iteration on the happy path:

- CryptnoxWallet::connect opens the channel,
- CryptnoxWallet::verifyPin sends DEMO_PIN,
- CryptnoxWallet::disconnect tears the channel down.

If CryptnoxWallet::verifyPin returns `false` the sketch enters an infinite halt. This is deliberate: looping would resubmit the same wrong PIN on every iteration and exhaust the card's retry counter within a few seconds, permanently blocking the PIN.
The 1 s delay between successful iterations gives time to remove/replace the card and keeps the Serial output readable.
Definition at line 97 of file VerifyPin.ino.
References DEMO_PIN, F, serialAdapter, and wallet.

6.19.2.2 nfc()

```
PN532Adapter nfc (
            serialAdapter ,
            PN532_SS_PIN ,
            & SPI)
```

PN532 transport adapter over SPI.
References PN532_SS_PIN, and serialAdapter.

6.19.2.3 setup()

```
void setup ()
```

Arduino setup hook.
Brings up `Serial`, the SPI bus and the PN532 reader. Halts on init failure (no reader detected) so the user can inspect the Serial output.
Definition at line 70 of file VerifyPin.ino.
References F, serialAdapter, and wallet.

6.19.3 Variable Documentation

6.19.3.1 cryptoProvider

`ArduinoCryptoProvider cryptoProvider`
Crypto provider (AES / SHA / micro-ecc / TRNG bridge for Arduino).
Definition at line 56 of file VerifyPin.ino.

6.19.3.2 platform

`ArduinoPlatform platform`
Platform adapter (Arduino blocking delay).
Definition at line 59 of file VerifyPin.ino.

6.19.3.3 serialAdapter

`ArduinoLoggerAdapter serialAdapter`
Arduino logger adapter — emits SDK diagnostics on `Serial`.
Definition at line 50 of file VerifyPin.ino.

6.19.3.4 wallet

```
CryptnoxWallet wallet(nfc, serialAdapter, cryptoProvider, platform) (
             nfc ,
             serialAdapter ,
             cryptoProvider ,
             platform )
```

High-level Cryptnox wallet wiring the four adapters together.

6.20 VerifyPin.ino

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
00037 #include <CryptnoxWallet.h>
00038 #include <SPI.h>
00039
00041 #define PN532_SS_PIN  (10U)
00042
00047 #define DEMO_PIN      "000000000"
00048
00050 ArduinoLoggerAdapter   serialAdapter;
00051
00053 PN532Adapter           nfc(serialAdapter, PN532_SS_PIN, &SPI);
00054
00056 ArduinoCryptoProvider  cryptoProvider;
00057
00059 ArduinoPlatform        platform;
00060
00062 CryptnoxWallet         wallet(nfc, serialAdapter, cryptoProvider, platform);
00063
00070 void setup() {
00071     serialAdapter.begin(115200);
00072     delay(1000);                              /* Arduino R4: wait for Serial */
00073     SPI.begin();
00074
00075     if (!wallet.begin()) {
00076         serialAdapter.println(F("PN532 init failed"));
00077         while (1);
00078     }
00079 }
00080
00097 void loop() {
00098     CW_SecureSession session;
00099     if (!wallet.connect(session)) {
00100         serialAdapter.println(F("Card not detected"));
00101         wallet.disconnect(session);
00102         delay(1000);
00103         return;
```

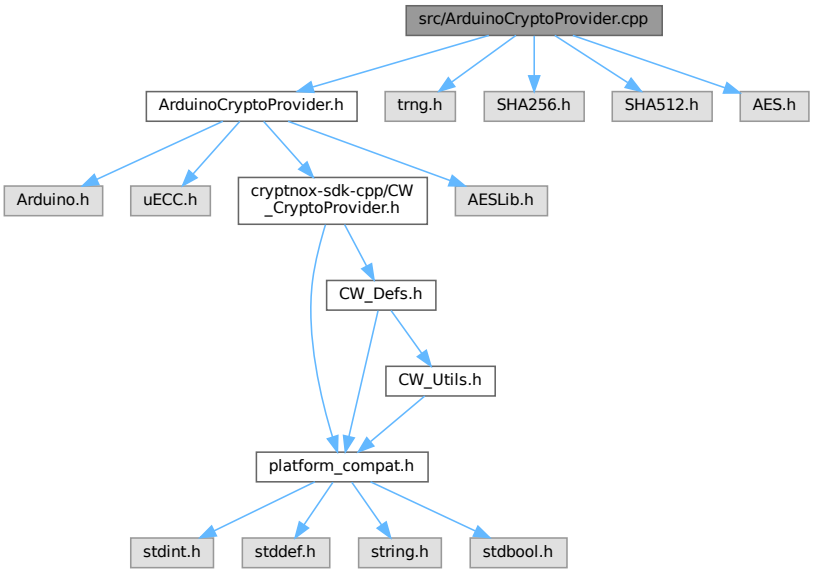
```
00104     }
00105
00106     if (!wallet.verifyPin(session,
00107                           reinterpret_cast<const uint8_t*>(DEMO_PIN),
00108                           (uint8_t)strlen(DEMO_PIN))) {
00109         /* CRITICAL: do NOT loop on failure -- each wrong PIN attempt
00110          * decrements the card's retry counter (typically 3-5 tries) and
00111          * permanently blocks the PIN at zero. Halt and let the developer
00112          * inspect the Serial output / fix DEMO_PIN before retrying. */
00113         serialAdapter.println(F("PIN rejected -- halting to protect retry counter"));
00114         wallet.disconnect(session);
00115         while (1);
00116     }
00117
00118     serialAdapter.println(F("PIN accepted"));
00119     wallet.disconnect(session);
00120     delay(1000);
00121 }
```

6.21 src/ArduinoCryptoProvider.cpp File Reference

Implementation of the Arduino UNO R4 concrete crypto provider.

```
#include "ArduinoCryptoProvider.h"
#include <trng.h>
#include <SHA256.h>
#include <SHA512.h>
#include <AES.h>
```

Include dependency graph for ArduinoCryptoProvider.cpp:



6.21.1 Detailed Description

Implementation of the Arduino UNO R4 concrete crypto provider.
Stitches together the four crypto libraries listed in the header (AESLib / Crypto / micro-ecc / trng) and forwards each CW_CryptoProvider operation to the appropriate backend. Full API documentation lives on the declarations in ArduinoCryptoProvider.h.
Definition in file ArduinoCryptoProvider.cpp.

6.22 ArduinoCryptoProvider.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
00015
00016 #include "ArduinoCryptoProvider.h"
00017 #include <trng.h>
00018 #if CW_VERIFY_CERT
00019 #include <SHA256.h>
00020 #endif
00021 #include <SHA512.h>
00022 #include <AES.h>
00023
00027 ArduinoCryptoProvider::ArduinoCryptoProvider() {
00028     uECC_set_rng(&ArduinoCryptoProvider::trngCallback);
00029 }
00030
00041 const uECC_Curve_t* ArduinoCryptoProvider::toUEccCurve(CW_Curve curve) {
00042     switch (curve) {
00043         case CW_CURVE_SECP256R1: return uECC_secp256r1();
00044 #if uECC_SUPPORTS_secp256k1
00045         case CW_CURVE_SECP256K1: return uECC_secp256k1();
00046 #endif
00047         default:                 return NULL;
00048     }
00049 }
00050
00054 uint8_t ArduinoCryptoProvider::trngByte() {
00055     /* Lazy init: the RA4M1 TRNG peripheral is enabled on first read so
00056      * the constructor stays cheap and order-of-construction-safe with
00057      * other global objects in the sketch. */
00058     static bool initialized = false;
00059     if (!initialized) {
00060         TRNG.begin();
00061         initialized = true;
00062     }
00063     uint8_t out = 0U;
00064     TRNG.random8(&out);
00065     return out;
00066 }
00067
00071 int ArduinoCryptoProvider::trngCallback(uint8_t* dest, unsigned size) {
00072     int ret = 0;
00073     if ((dest != NULL) && (size > 0U)) {
00074         for (unsigned i = 0U; i < size; i++) {
00075             dest[i] = trngByte();
00076         }
00077         ret = 1;
00078     }
00079     return ret;
00080 }
00081
00085 bool ArduinoCryptoProvider::sha256(const uint8_t* data, size_t len, uint8_t* out) {
00086     if (out == NULL) { return false; }
00087 #if CW_VERIFY_CERT
00088     if ((data == NULL) && (len > 0U)) { return false; }
00089     SHA256 sha;
00090     sha.update(data, len);
00091     sha.finalize(out, 32U);
00092 #else
00093     /* Card-certificate verification compiled out: keep the symbol but skip
00094      * the work so the SHA256 dependency is not pulled in by the linker. */
00095     (void)data; (void)len; (void)out;
00096 #endif
00097     return true;
00098 }
00099
00103 bool ArduinoCryptoProvider::sha512(const uint8_t* data, size_t len, uint8_t* out) {
00104     if (out == NULL) { return false; }
00105     if ((data == NULL) && (len > 0U)) { return false; }
00106     SHA512 sha;
00107     sha.update(data, len);
00108     sha.finalize(out, 64U);
00109     return true;
00110 }
00111
00115 uint16_t ArduinoCryptoProvider::aesCbcEncrypt(const uint8_t* in, uint16_t len, uint8_t* out,
00116                                               const uint8_t* key, uint8_t keyLen,
00117                                               uint8_t* iv, bool bitPadding) {
00118     _aes.set_paddingmode(bitPadding ? paddingMode::Bit : paddingMode::Null);
00119     return _aes.encrypt(reinterpret_cast<const byte*>(in), len, out,
00120                         reinterpret_cast<const byte*>(key), static_cast<int>(keyLen), iv);
```

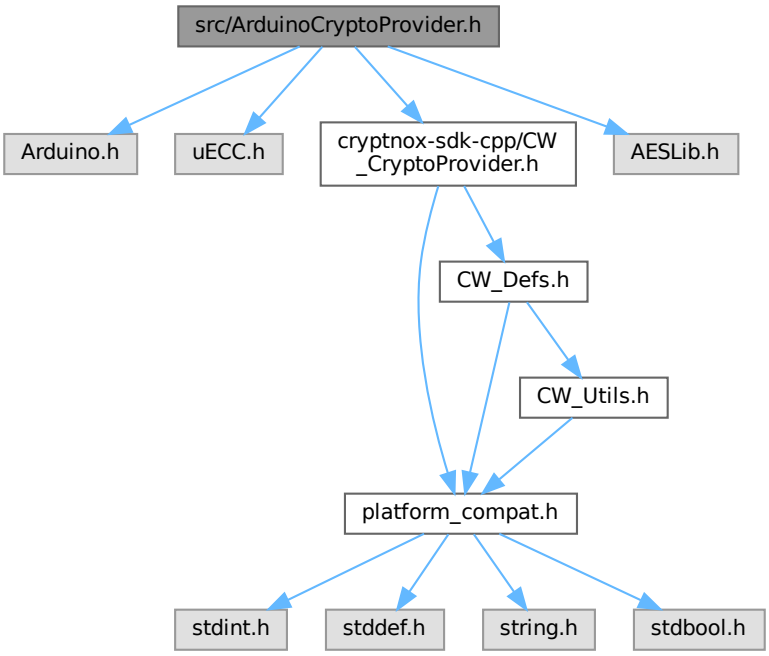
```
00121 }
00122
00126 uint16_t ArduinoCryptoProvider::aesCbcDecrypt(uint8_t* in, uint16_t len, uint8_t* out,
00127                                                const uint8_t* key, uint8_t keyLen,
00128                                                uint8_t* iv, bool bitPadding) {
00129     _aes.set_paddingmode(bitPadding ? paddingMode::Bit : paddingMode::Null);
00130     return _aes.decrypt(in, len, out,
00131                         reinterpret_cast<const byte*>(key), static_cast<int>(keyLen), iv);
00132 }
00133
00137 bool ArduinoCryptoProvider::ecdh(const uint8_t* pubKey, const uint8_t* privKey,
00138                                  uint8_t* secret, CW_Curve curve) {
00139     const uECC_Curve_t* uc = toUEccCurve(curve);
00140     if (uc == NULL) { return false; }
00141     return (uECC_shared_secret(pubKey, privKey, secret, uc) != 0);
00142 }
00143
00147 bool ArduinoCryptoProvider::makeKey(uint8_t* pubKey, uint8_t* privKey,
00148                                     CW_Curve curve) {
00149     const uECC_Curve_t* uc = toUEccCurve(curve);
00150     if (uc == NULL) { return false; }
00151     return (uECC_make_key(pubKey, privKey, uc) != 0);
00152 }
00153
00157 bool ArduinoCryptoProvider::random(uint8_t* dest, unsigned size) {
00158     return (trngCallback(dest, size) == 1);
00159 }
00160
00164 bool ArduinoCryptoProvider::ecdsaVerify(const uint8_t* pubKey64,
00165                                         const uint8_t* hash, size_t hashLen,
00166                                         const uint8_t* sig, CW_Curve curve) {
00167     const uECC_Curve_t* uc = toUEccCurve(curve);
00168     if (uc == NULL) { return false; }
00169     return (uECC_verify(pubKey64, hash, static_cast<unsigned>(hashLen), sig, uc) != 0);
00170 }
```

6.23 src/ArduinoCryptoProvider.h File Reference

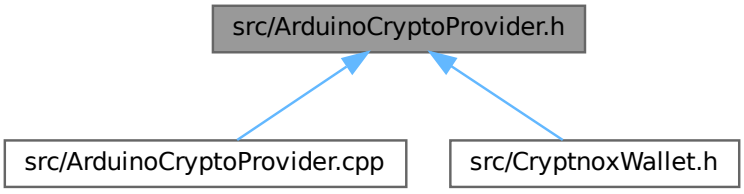
Concrete `CW_CryptoProvider` for the Arduino UNO R4 (RA4M1).

```
#include <Arduino.h>
#include <uECC.h>
#include "cryptnox-sdk-cpp/CW_CryptoProvider.h"
#include "AESLib.h"
```

Include dependency graph for ArduinoCryptoProvider.h:



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



Classes

- class ArduinoCryptoProvider

  CW_CryptoProvider implementation for the Arduino UNO R4 (RA4M1).

6.23.1 Detailed Description

Concrete CW_CryptoProvider for the Arduino UNO R4 (RA4M1).
Declares ArduinoCryptoProvider, the Arduino-side concrete crypto adapter the host integration injects into CryptnoxWallet. The adapter stitches together four independent libraries — AESLib, the rweather Crypto SHA-256/512 implementations, micro-ecc, and the on-chip RA4M1 TRNG — behind the single platform-independent CW_CryptoProvider interface.

See also

CW_CryptoProvider

CryptnoxWallet

Definition in file ArduinoCryptoProvider.h.

6.24 ArduinoCryptoProvider.h

Go to the documentation of this file.

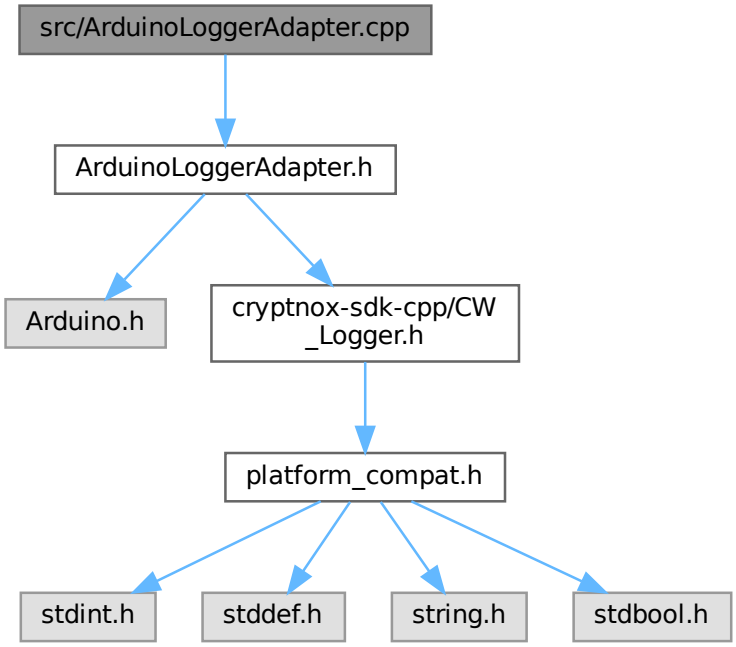
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00020
00021 #ifndef ARDUINOCRYPTOPROVIDER_H
00022 #define ARDUINOCRYPTOPROVIDER_H
00023
00024 #include <Arduino.h>
00025 #include <uECC.h>
00026 #include "cryptnox-sdk-cpp/CW_CryptoProvider.h"
00027 #include "AESLib.h"
00028
00063 class ArduinoCryptoProvider : public CW_CryptoProvider {
00064 public:
00068     ArduinoCryptoProvider();
00069
00070     ArduinoCryptoProvider(const ArduinoCryptoProvider&) = delete;
00071     ArduinoCryptoProvider& operator=(const ArduinoCryptoProvider&) = delete;
00072
00075
00089     bool sha256(const uint8_t* data, size_t len, uint8_t* out) override;
00090
00102     bool sha512(const uint8_t* data, size_t len, uint8_t* out) override;
00103
00117     uint16_t aesCbcEncrypt(const uint8_t* in, uint16_t len, uint8_t* out,
00118                            const uint8_t* key, uint8_t keyLen,
00119                            uint8_t* iv, bool bitPadding) override;
00120
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
00170     bool makeKey(uint8_t* pubKey, uint8_t* privKey,
00171                  CW_Curve curve) override;
00172
00183     bool random(uint8_t* dest, unsigned size) override;
00184
00199     bool ecdsaVerify(const uint8_t* pubKey64,
00200                      const uint8_t* hash, size_t hashLen,
00201                      const uint8_t* sig, CW_Curve curve) override;
00203
00204 private:
00205     AESLib _aes;
00206
00213     static const uECC_Curve_t* toUEccCurve(CW_Curve curve);
00214
00222     static uint8_t trngByte();
00223
00231     static int trngCallback(uint8_t* dest, unsigned size);
00232 };
00233
00234 #endif // ARDUINOCRYPTOPROVIDER_H
```

6.25 src/ArduinoLoggerAdapter.cpp File Reference

Implementation of the HardwareSerial-backed logger.

```
#include "ArduinoLoggerAdapter.h"
```

Include dependency graph for ArduinoLoggerAdapter.cpp:



6.25.1 Detailed Description

Implementation of the HardwareSerial-backed logger.
Each method is a one-line forward to the wrapped `HardwareSerial` instance. Doxygen documentation lives on the declarations in ArduinoLoggerAdapter.h.
Definition in file ArduinoLoggerAdapter.cpp.

6.26 ArduinoLoggerAdapter.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 "ArduinoLoggerAdapter.h"
00016
00017 ArduinoLoggerAdapter::ArduinoLoggerAdapter()
00018     : _serial(&Serial) {
00019 }
00020
00021 ArduinoLoggerAdapter::ArduinoLoggerAdapter(HardwareSerial* serial)
00022     : _serial(serial) {
00023 }
00024
00025 bool ArduinoLoggerAdapter::begin(unsigned long baudRate) {
00026     _serial->begin(baudRate);
00027     return true;
00028 }
00029
00030 void ArduinoLoggerAdapter::print(const __FlashStringHelper* str) { _serial->print(str); }
00031 void ArduinoLoggerAdapter::print(const char* str)                 { _serial->print(str); }
00032 void ArduinoLoggerAdapter::print(char c)                          { _serial->print(c); }
00033 void ArduinoLoggerAdapter::print(uint8_t value, int base)         { _serial->print(value, base); }
```

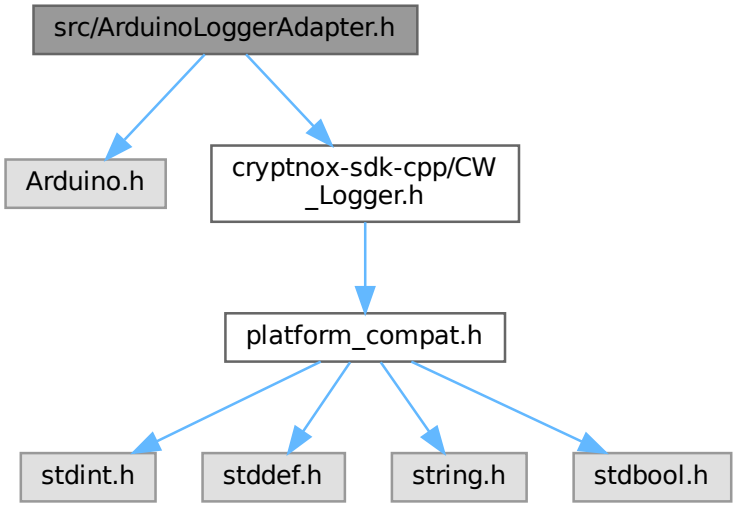
```
00034 void ArduinoLoggerAdapter::print(uint16_t value, int base)        { _serial->print(value, base); }
00035 void ArduinoLoggerAdapter::print(uint32_t value, int base)        { _serial->print(value, base); }
00036 void ArduinoLoggerAdapter::print(int value, int base)             { _serial->print(value, base); }
00037
00038 void ArduinoLoggerAdapter::println()                               { _serial->println(); }
00039 void ArduinoLoggerAdapter::println(const __FlashStringHelper* str)  { _serial->println(str); }
00040 void ArduinoLoggerAdapter::println(const char* str)                { _serial->println(str); }
00041 void ArduinoLoggerAdapter::println(char c)                         { _serial->println(c); }
00042 void ArduinoLoggerAdapter::println(uint8_t value, int base)        { _serial->println(value, base); }
00043 void ArduinoLoggerAdapter::println(uint16_t value, int base)       { _serial->println(value, base); }
00044 void ArduinoLoggerAdapter::println(uint32_t value, int base)       { _serial->println(value, base); }
00045 void ArduinoLoggerAdapter::println(int value, int base)            { _serial->println(value, base); }
```

6.27 src/ArduinoLoggerAdapter.h File Reference

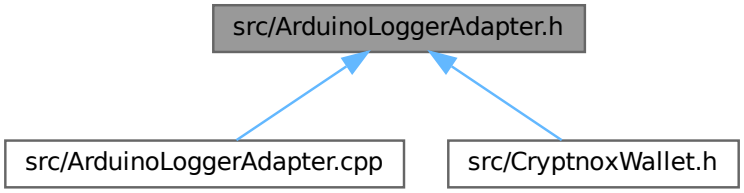
Concrete `CW_Logger` over `HardwareSerial` for development builds.

```
#include <Arduino.h>
#include "cryptnox-sdk-cpp/CW_Logger.h"
```

Include dependency graph for ArduinoLoggerAdapter.h:



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



Classes

- class ArduinoLoggerAdapter

  `CW_Logger` implementation wrapping Arduino's `HardwareSerial`.

6.27.1 Detailed Description

Concrete `CW_Logger` over `HardwareSerial` for development builds.
Declares ArduinoLoggerAdapter, a thin pass-through adapter that sends every log call to an Arduino `HardwareSerial` (defaults to the primary `Serial`). Intended for development and debugging.

See also

CW_Logger

NullLoggerAdapter — production-time replacement that silences output.

Definition in file ArduinoLoggerAdapter.h.

6.28 ArduinoLoggerAdapter.h

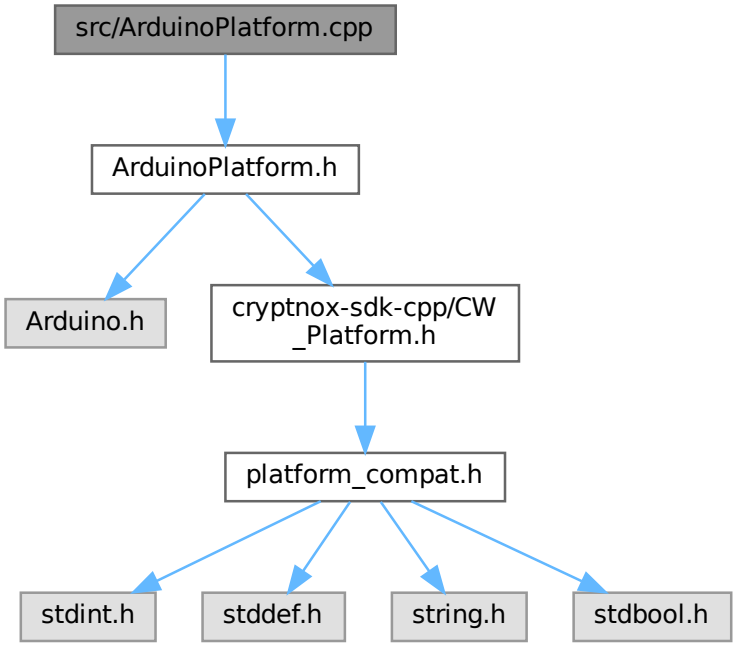
Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00017
00018 #ifndef ARDUINOLOGGERADAPTER_H
00019 #define ARDUINOLOGGERADAPTER_H
00020
00021 #include <Arduino.h>
00022 #include "cryptnox-sdk-cpp/CW_Logger.h"
00023
00048 class ArduinoLoggerAdapter : public CW_Logger {
00049 public:
00057     ArduinoLoggerAdapter();
00058
00066     explicit ArduinoLoggerAdapter(HardwareSerial* serial);
00067
00068     ~ArduinoLoggerAdapter() override = default;
00069
00070     ArduinoLoggerAdapter(const ArduinoLoggerAdapter&) = delete;
00071     ArduinoLoggerAdapter& operator=(const ArduinoLoggerAdapter&) = delete;
00072
00075
00086     bool begin(unsigned long baudRate = 115200UL) override;
00087
00088     void print(const __FlashStringHelper* str) override;
00089     void print(const char* str) override;
00090     void print(char c) override;
00091     void print(uint8_t value, int base = DEC) override;
00092     void print(uint16_t value, int base = DEC) override;
00093     void print(uint32_t value, int base = DEC) override;
00094     void print(int value, int base = DEC) override;
00095
00096     void println() override;
00097     void println(const __FlashStringHelper* str) override;
00098     void println(const char* str) override;
00099     void println(char c) override;
00100     void println(uint8_t value, int base = DEC) override;
00101     void println(uint16_t value, int base = DEC) override;
00102     void println(uint32_t value, int base = DEC) override;
00103     void println(int value, int base = DEC) override;
00105
00106 private:
00107     HardwareSerial* _serial;
00108 };
00109
00110 #endif // ARDUINOLOGGERADAPTER_H
```

6.29 src/ArduinoPlatform.cpp File Reference

Implementation of the Arduino blocking-delay platform adapter.

`#include "ArduinoPlatform.h"`
Include dependency graph for ArduinoPlatform.cpp:



6.29.1 Detailed Description

Implementation of the Arduino blocking-delay platform adapter.
Definition in file ArduinoPlatform.cpp.

6.30 ArduinoPlatform.cpp

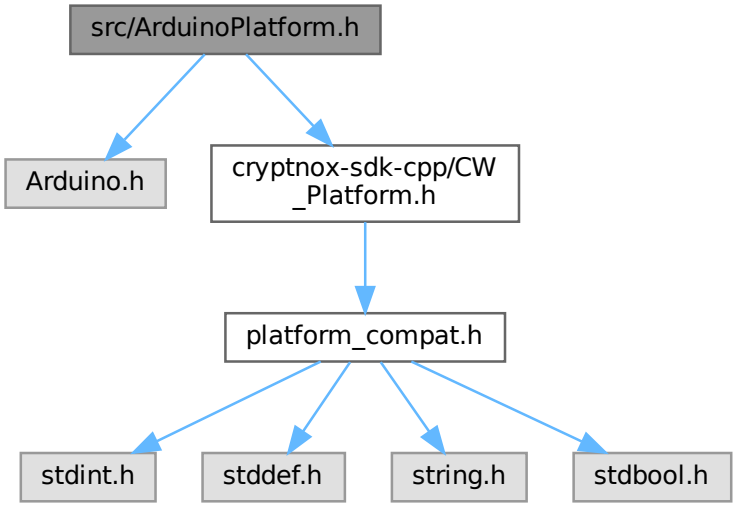
Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00010
00011 #include "ArduinoPlatform.h"
00012
00016 void ArduinoPlatform::sleep_ms(uint32_t ms) {
00017     delay(ms);
00018 }
```

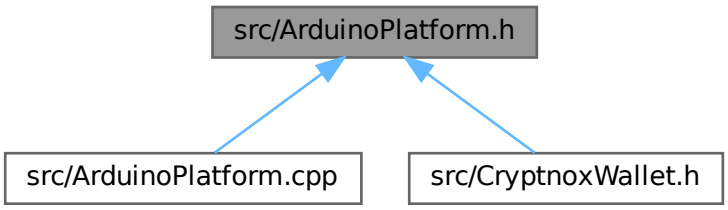
6.31 src/ArduinoPlatform.h File Reference

Concrete `CW_Platform` over Arduino's `delay()`.
`#include <Arduino.h>`
`#include "cryptnox-sdk-cpp/CW_Platform.h"`

Include dependency graph for ArduinoPlatform.h:



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



Classes

- class ArduinoPlatform

  `CW_Platform` *implementation using Arduino's blocking* `delay()`.

6.31.1 Detailed Description

Concrete `CW_Platform` over Arduino's `delay()`.
Declares ArduinoPlatform, the Arduino-side concrete platform adapter the host integration injects into CryptnoxWallet. The SDK uses the single CW_Platform::sleep_ms primitive to space APDU exchanges across the secure-channel handshake; on Arduino that maps directly to the framework's blocking `delay()` call.

See also

CW_Platform

CryptnoxWallet

Definition in file ArduinoPlatform.h.

6.32 ArduinoPlatform.h

Go to the documentation of this file.

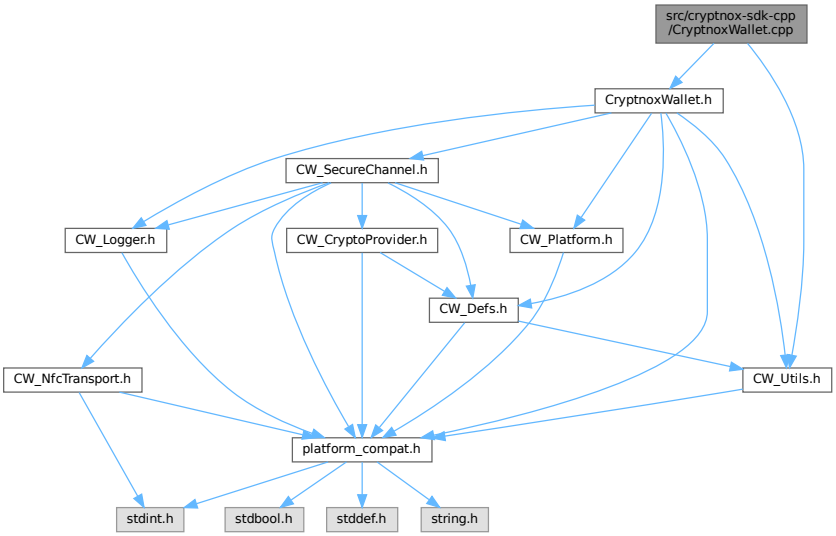
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00019
00020 #ifndef ARDUINOPLATFORM_H
00021 #define ARDUINOPLATFORM_H
00022
00023 #include <Arduino.h>
00024 #include "cryptnox-sdk-cpp/CW_Platform.h"
00025
00049 class ArduinoPlatform : public CW_Platform {
00050 public:
00051     ArduinoPlatform() = default;
00052     ~ArduinoPlatform() override = default;
00053
00054     ArduinoPlatform(const ArduinoPlatform&) = delete;
00055     ArduinoPlatform& operator=(const ArduinoPlatform&) = delete;
00056
00062     void sleep_ms(uint32_t ms) override;
00063 };
00064
00065 #endif // ARDUINOPLATFORM_H
```

6.33 src/cryptnox-sdk-cpp/CryptnoxWallet.cpp File Reference

Implementation of the high-level CryptnoxWallet API.

```
#include "CryptnoxWallet.h"
#include "CW_Utils.h"
```

Include dependency graph for CryptnoxWallet.cpp:



6.33.1 Detailed Description

Implementation of the high-level CryptnoxWallet API.
Coordinates the secure channel layer (CW_SecureChannel) with the higher-level wallet operations declared in CryptnoxWallet.h. Handles connection retries, sensitive buffer wiping on every exit path, PIN/sign payload assembly, and DER signature parsing.
Definition in file CryptnoxWallet.cpp.

6.34 CryptnoxWallet.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
00015
00016 /* NOTE: Do NOT include <Arduino.h> here -- this is a platform-independent file.
00017  * Arduino compatibility shims (F(), HEX, delay) are provided via
00018  * platform_compat.h which is pulled in transitively through CryptnoxWallet.h. */
00019 #include "CryptnoxWallet.h"
00020 #include "CW_Utils.h"
00021
00022 /**************************************************************************
00023  * Constructor
00024  **************************************************************************/
00025
00026 // cppcheck-suppress misra-c2012-12.3 -- C++: member initializer-list commas are not the comma
      operator
00027 CryptnoxWallet::CryptnoxWallet(CW_NfcTransport& driver, CW_Logger& logger,
00028                                CW_CryptoProvider& crypto, CW_Platform& platform)
00029     : _logger(logger), _platform(platform), _secure(driver, logger, crypto, platform) {
00030 }
00031
00032 /**************************************************************************
00033  * Public methods
00034  **************************************************************************/
00035
00036 bool CryptnoxWallet::begin() {
00037     bool ret = _secure.begin();
00038     if (ret) {
00039         printPN532FirmwareVersion();
00040     }
00041     return ret;
00042 }
00043
00044 bool CryptnoxWallet::connect(CW_SecureSession& session) {
00045     bool ret = false;
00046     session.clear(); /* CRIT-04: clear any stale keys from a previous or partial session */
00047
00048     for (uint8_t attempt = 0U; (attempt < CW_CONNECT_MAX_ATTEMPTS) && (ret == false); attempt++) {
00049         if (attempt > 0U) {
00050             session.clear(); /* CRIT-04: clear partial keys left by a failed attempt before retrying
      */
00051 #if CW_DEBUG_LOGGING
00052             _logger.print(F("Retrying card connection (attempt "));
00053             _logger.print((uint8_t)(attempt + 1U));
00054             _logger.println(F(")..."));
00055 #endif
00056             _secure.resetReader();
00057             _platform.sleep_ms(200U);
00058         }
00059
00060         if (_secure.inListPassiveTarget()) {
00061             _platform.sleep_ms(200U);
00062             if (establishSecureChannel(session)) {
00063                 ret = true;
00064             }
00065         }
00066     }
00067
00068     if (!ret) {
00069         session.clear(); /* CRIT-04: clear any partial keys from the final failed attempt */
00070     }
00071
00072     return ret;
00073 }
00074
00075 bool CryptnoxWallet::establishSecureChannel(CW_SecureSession& session) {
00076     bool ret = false;
00077
```

```
00078     /* Declare all sensitive stack buffers at function entry so they can be
00079      * wiped on every exit path (H-01, M-02). */
00080     uint8_t cardCertificate[146U]      = { 0U };
00081     uint8_t cardCertificateLength      = 0U;
00082     uint8_t cardEphemeralPubKey[64U]   = { 0U };
00083     uint8_t openSecureChannelSalt[32U] = { 0U };
00084     uint8_t clientPrivateKey[32U]      = { 0U };
00085     uint8_t clientPublicKey[64U]       = { 0U };
00086     CW_Curve sessionCurve              = CW_CURVE_SECP256R1;
00087
00088     if (_secure.selectApdu()) {
00089         /* Fetch the manufacturer certificate BEFORE getCardCertificate().
00090          * The Cryptnox card state machine advances after GET_CARD_CERTIFICATE
00091          * (INS=F8) and will not respond to GET_MANUFACTURER_CERTIFICATE (INS=F7)
00092          * after that point.  Pre-fetching here caches the cert inside
00093          * CW_SecureChannel so that verifyCertificateChain() can use it without
00094          * issuing another APDU. */
00095         if (!_secure.preFetchManufacturerCert()) {
00096 #if CW_DEBUG_LOGGING
00097             _logger.println(F("Failed to pre-fetch manufacturer certificate"));
00098 #endif
00099         } else {
00100             if (_secure.getCardCertificate(cardCertificate, cardCertificateLength)) {
00101                 uint8_t certResult = _secure.verifyCertificateChain(cardCertificate,
00102                                                                     cardCertificateLength);
00103                 if (certResult != CW_CERT_OK) {
00104 #if CW_DEBUG_LOGGING
00105                     _logger.print(F("Card authenticity check failed (code 0x"));
00106                     _logger.print(certResult, HEX);
00107                     _logger.println(F("). Aborting."));
00108 #endif
00109                 } else {
00110                     if (_secure.extractCardEphemeralKey(cardCertificate, cardEphemeralPubKey)) {
00111                         if (_secure.openSecureChannel(openSecureChannelSalt, clientPublicKey,
00112                                                       clientPrivateKey, sessionCurve)) {
00113                             if (_secure.mutuallyAuthenticate(session, openSecureChannelSalt,
00114                                                              clientPublicKey, clientPrivateKey,
00115                                                              sessionCurve, cardEphemeralPubKey)) {
00116 #if CW_DEBUG_LOGGING
00117                                 _logger.println(F("Secure channel established"));
00118 #endif
00119                                 ret = true;
00120                             } else {
00121 #if CW_DEBUG_LOGGING
00122                                 _logger.println(F("Mutual authentication failed"));
00123 #endif
00124                             }
00125                         } else {
00126 #if CW_DEBUG_LOGGING
00127                             _logger.println(F("Failed to open secure channel"));
00128 #endif
00129                         }
00130                     } else {
00131 #if CW_DEBUG_LOGGING
00132                         _logger.println(F("Failed to extract card ephemeral key"));
00133 #endif
00134                     }
00135                 }
00136             } else {
00137 #if CW_DEBUG_LOGGING
00138                 _logger.println(F("Failed to get card certificate"));
00139 #endif
00140             }
00141         } /* end preFetchManufacturerCert else */
00142     } else {
00143 #if CW_DEBUG_LOGGING
00144         _logger.println(F("Failed to select Cryptnox application"));
00145 #endif
00146     }
00147
00148     /* Wipe all sensitive ephemeral key material on every exit path (H-01, M-02). */
00149     CW_Utils::secure_wipe(clientPrivateKey,      sizeof(clientPrivateKey));
00150     CW_Utils::secure_wipe(openSecureChannelSalt, sizeof(openSecureChannelSalt));
00151     CW_Utils::secure_wipe(clientPublicKey,       sizeof(clientPublicKey));
00152     CW_Utils::secure_wipe(cardEphemeralPubKey,   sizeof(cardEphemeralPubKey));
00153     CW_Utils::secure_wipe(cardCertificate,       sizeof(cardCertificate));
00154
00155     return ret;
00156 }
00157
00158 void CryptnoxWallet::disconnect(CW_SecureSession& session) {
00159     if (isSecureChannelOpen(session)) {
00160         session.clear();
00161     }
00162     _secure.resetReader();
00163 }
00164
```

```
00165 bool CryptnoxWallet::getCardInfo(CW_SecureSession& session, CW_CardInfo* info) {
00166     bool ret = false;
00167     if (!isSecureChannelOpen(session)) {
00168 #if CW_DEBUG_LOGGING
00169         _logger.println(F("Error: Secure channel not open. Cannot get card info."));
00170 #endif
00171         return false;
00172     }
00173     uint8_t data[] = { 0x00U };
00174     uint8_t apdu[] = { 0x80U, 0xFAU, 0x00U, 0x00U };
00175
00176     uint8_t  decrypted[255U] = { 0U };
00177     uint16_t decryptedLen    = 0U;
00178
00179     ret = _secure.aesCbcEncrypt(session, apdu, sizeof(apdu),
00180                                 data, sizeof(data),
00181                                 decrypted, &decryptedLen);
00182
00183     if (ret && (info != NULL)) {
00184         /* Response layout (Cryptnox basic_g1 spec):
00185          *   [byte0] [name_len(1)] [name(name_len)]
00186          *           [email_len(1)] [email(email_len)] [... more fields ...]
00187          * byte0 = unused/flags. */
00188         ret = false;
00189         if (decryptedLen >= 4U) {
00190             uint16_t pos     = 1U;
00191             uint8_t  nameLen = decrypted[pos];
00192             pos += 1U;
00193             if ((nameLen <= CW_CARD_NAME_MAX_LEN) &&
00194                 ((uint16_t)(pos + nameLen + 1U) <= decryptedLen)) {
00195                 (void)CW_Utils::safe_memcpy(reinterpret_cast<uint8_t*>(info->name),
00196                                             sizeof(info->name),
00197                                             decrypted + pos, nameLen);
00198                 info->name[nameLen] = '\0';
00199                 pos += nameLen;
00200
00201                 uint8_t emailLen = decrypted[pos];
00202                 pos += 1U;
00203                 if ((emailLen <= CW_CARD_EMAIL_MAX_LEN) &&
00204                     ((uint16_t)(pos + emailLen) <= decryptedLen)) {
00205                     (void)CW_Utils::safe_memcpy(reinterpret_cast<uint8_t*>(info->email),
00206                                                 sizeof(info->email),
00207                                                 decrypted + pos, emailLen);
00208                     info->email[emailLen] = '\0';
00209                     ret = true;
00210                 }
00211             }
00212         }
00213     }
00214
00215     CW_Utils::secure_wipe(decrypted, sizeof(decrypted));
00216     return ret;
00217 }
00218
00219 bool CryptnoxWallet::verifyPin(CW_SecureSession& session, const uint8_t* pin, uint8_t pinLength) {
00220     bool ret = false;
00221     if (!isSecureChannelOpen(session)) {
00222 #if CW_DEBUG_LOGGING
00223         _logger.println(F("Error: Secure channel not open. Cannot verify PIN."));
00224 #endif
00225     }
00226     else if ((pin == NULL) || (pinLength < CW_MIN_PIN_LENGTH) || (pinLength > CW_MAX_PIN_LENGTH)) {
00227 #if CW_DEBUG_LOGGING
00228         _logger.println(F("Error: Invalid PIN (must be 4-9 digits)."));
00229 #endif
00230     }
00231     else {
00232         uint8_t paddedPin[CW_MAX_PIN_LENGTH] = { 0U };
00233         (void)CW_Utils::safe_memcpy(paddedPin, sizeof(paddedPin), pin, pinLength);
00234         uint8_t apdu[] = { 0x80U, 0x20U, 0x00U, 0x00U };
00235         ret = _secure.aesCbcEncrypt(session, apdu, sizeof(apdu), paddedPin, CW_MAX_PIN_LENGTH);
00236         CW_Utils::secure_wipe(paddedPin, sizeof(paddedPin));
00237     }
00238     return ret;
00239 }
00240
00241 bool CryptnoxWallet::writeUserData(CW_SecureSession& session, uint8_t slot,
00242                                    const uint8_t* data, uint16_t dataLength) {
00243     bool ret = false;
00244
00245     if (!isSecureChannelOpen(session)) {
00246 #if CW_DEBUG_LOGGING
00247         _logger.println(F("Error: Secure channel not open. Cannot write user data."));
00248 #endif
00249     }
00250     else if ((data == NULL) || (dataLength == 0U)) {
00251 #if CW_DEBUG_LOGGING
```

```
00252         _logger.println(F("Error: Invalid data for write user data."));
00253 #endif
00254     }
00255     else {
00256         uint16_t offset = 0U;
00257         uint8_t  page   = 0U;
00258         ret = true;
00259 
00260         while ((offset < dataLength) && ret) {
00261             uint16_t chunkSize = dataLength - offset;
00262             if (chunkSize > CW_USER_DATA_PAGE_SIZE) {
00263                 chunkSize = CW_USER_DATA_PAGE_SIZE;
00264             }
00265 
00266             uint8_t apdu[] = { 0x80U, 0xFCU, slot, page };
00267 
00268 #if CW_DEBUG_LOGGING
00269             _logger.print(F("Writing user data page "));
00270             _logger.print(page);
00271             _logger.print(F(" ("));
00272             _logger.print(chunkSize);
00273             _logger.println(F(" bytes)..."));
00274 #endif
00275 
00276             if (!_secure.aesCbcEncrypt(session, apdu, sizeof(apdu), data + offset, chunkSize)) {
00277 #if CW_DEBUG_LOGGING
00278                 _logger.print(F("Error: Write user data failed on page "));
00279                 _logger.println(page);
00280 #endif
00281                 ret = false;
00282             }
00283             else {
00284                 offset += chunkSize;
00285                 page++;
00286             }
00287         }
00288     }
00289 
00290     return ret;
00291 }
00292 
00293 CW_SignResult CryptnoxWallet::sign(CW_SignRequest& request) {
00294     CW_SignResult result;
00295 
00296     if (validateSignRequest(request, result)) {
00297         uint8_t data[CW_HASH_SIZE + CW_MAX_DERIVE_PATH_LENGTH + CW_MAX_PIN_LENGTH] = { 0U };
00298         uint16_t dataLength = 0U;
00299 
00300         buildSignPayload(request, data, dataLength);
00301 
00302         uint8_t derResponse[255U] = { 0U };
00303         uint16_t derLength = 0U;
00304 
00305         if (sendSignApdu(request, data, dataLength, derResponse, derLength, result)) {
00306             if (extractRawSignature(derResponse, derLength, result)) {
00307                 debugPrintSignature(result.signature);
00308                 result.errorCode = CW_OK;
00309             }
00310         }
00311         CW_Utils::secure_wipe(data, sizeof(data));
00312         CW_Utils::secure_wipe(derResponse, sizeof(derResponse));
00313     }
00314 
00315     return result;
00316 }
00317 
00318 /**************************************************************************
00319  * Static public methods
00320  **************************************************************************/
00321 
00322 bool CryptnoxWallet::parseDerSignature(const uint8_t* der, uint8_t derLength,
00323                                        uint8_t* r, uint8_t& rLength,
00324                                        uint8_t* s, uint8_t& sLength) {
00325     bool ret = false;
00326 
00327     if ((der == NULL) || (derLength < 6U) || (r == NULL) || (s == NULL)) {
00328     }
00329     else if (der[0] != CW_DER_TAG_SEQUENCE) {
00330     }
00331     else {
00332         uint8_t pos = 2U;
00333 
00334         if (der[pos] != CW_DER_TAG_INTEGER) {
00335         }
00336         else {
00337             pos++;
00338             rLength = der[pos];
```

```
00339             pos++;
00340             if ((rLength > 33U) || ((pos + rLength) > derLength)) {
00341             }
00342             else {
00343                 (void)CW_Utils::safe_memcpy(r, 33U, der + pos, rLength);
00344                 pos += rLength;
00345 
00346                 if ((pos >= derLength) || (der[pos] != CW_DER_TAG_INTEGER)) {
00347                 }
00348                 else {
00349                     pos++;
00350                     sLength = der[pos];
00351                     pos++;
00352                     if ((sLength > 33U) || ((pos + sLength) > derLength)) {
00353                     }
00354                     else {
00355                         (void)CW_Utils::safe_memcpy(s, 33U, der + pos, sLength);
00356                         ret = true;
00357                     }
00358                 }
00359             }
00360         }
00361     }
00362 
00363     return ret;
00364 }
00365 
00366 /**************************************************************
00367  * Private methods
00368  **************************************************************/
00369 
00370 bool CryptnoxWallet::isSecureChannelOpen(const CW_SecureSession& session) const {
00371     uint8_t acc = 0U;
00372     for (uint8_t i = 0U; i < CW_AESKEY_SIZE; i++) {
00373         acc |= session.aesKey[i];
00374     }
00375     return (acc != 0U);
00376 }
00377 
00378 bool CryptnoxWallet::printPN532FirmwareVersion() {
00379     return _secure.printFirmwareVersion();
00380 }
00381 
00382 bool CryptnoxWallet::validateSignRequest(const CW_SignRequest& request, CW_SignResult& result) {
00383     bool ret = false;
00384 
00385     if (!isSecureChannelOpen(request.session)) {
00386 #if CW_DEBUG_LOGGING
00387         _logger.println(F("Error: Secure channel not open. Cannot sign."));
00388 #endif
00389         result.errorCode = CW_INVALID_SESSION;
00390     }
00391     else if ((request.hash == NULL) || (request.hashLength == 0U)) {
00392 #if CW_DEBUG_LOGGING
00393         _logger.println(F("Error: Invalid parameters for sign."));
00394 #endif
00395         result.errorCode = CW_SIGN_KEY_TOO_SHORT;
00396     }
00397     else if (request.hashLength > CW_HASH_SIZE) {
00398 #if CW_DEBUG_LOGGING
00399         _logger.println(F("Error: Hash too large."));
00400 #endif
00401         result.errorCode = CW_SIGN_KEY_TOO_SHORT;
00402     }
00403     else if ((request.pinLessMode) && (request.keyType != CW_SIGN_PINLESS_K1)) {
00404 #if CW_DEBUG_LOGGING
00405         _logger.println(F("Error: PIN-less mode requires CW_SIGN_PINLESS_K1 key type."));
00406 #endif
00407         result.errorCode = CW_SIGN_KEY_TOO_SHORT_WITH_PINLESS_MODE;
00408     }
00409     else {
00410         ret = true;
00411 
00412         if (!request.pinLessMode) {
00413             uint8_t pinLength = 0U;
00414             for (uint8_t i = 0U; i < CW_MAX_PIN_LENGTH; i++) {
00415                 if (request.pin[i] == 0U) { break; }
00416                 pinLength++;
00417             }
00418             if ((pinLength > 0U) && (pinLength < CW_MIN_PIN_LENGTH)) {
00419 #if CW_DEBUG_LOGGING
00420                 _logger.println(F("Error: PIN too short (must be 4-9 digits)."));
00421 #endif
00422                 result.errorCode = CW_SIGN_PIN_INCORRECT;
00423                 ret = false;
00424             }
00425         }
```

```
00426     }
00427
00428     return ret;
00429 }
00430
00431 void CryptnoxWallet::buildSignPayload(const CW_SignRequest& request,
00432                                       uint8_t* data, uint16_t& dataLength) {
00433     const size_t kDataBufSize = static_cast<size_t>(CW_HASH_SIZE) +
      static_cast<size_t>(CW_MAX_DERIVE_PATH_LENGTH) + static_cast<size_t>(CW_MAX_PIN_LENGTH);
00434     dataLength = request.hashLength;
00435     (void)CW_Utils::safe_memcpy(data, kDataBufSize, request.hash, request.hashLength);
00436
00437     if ((request.keyType == CW_SIGN_DERIVE_K1 || request.keyType == CW_SIGN_DERIVE_R1) &&
00438         (request.derivePath != NULL) && (request.derivePathLength > 0U)) {
00439         (void)CW_Utils::safe_memcpy(data + dataLength, kDataBufSize - static_cast<size_t>(dataLength),
      request.derivePath, request.derivePathLength);
00440         dataLength += request.derivePathLength;
00441     }
00442
00443     if (!request.pinLessMode) {
00444         uint8_t pinLength = 0U;
00445         for (uint8_t i = 0U; i < CW_MAX_PIN_LENGTH; i++) {
00446             if (request.pin[i] == 0U) { break; }
00447             pinLength++;
00448         }
00449         if (pinLength > 0U) {
00450             (void)CW_Utils::safe_memcpy(data + dataLength, kDataBufSize -
      static_cast<size_t>(dataLength), request.pin, CW_MAX_PIN_LENGTH);
00451             dataLength += CW_MAX_PIN_LENGTH;
00452         }
00453     }
00454 }
00455
00456 bool CryptnoxWallet::sendSignApdu(CW_SignRequest& request, const uint8_t* data,
00457                                   uint16_t dataLength, uint8_t* derResponse,
00458                                   uint16_t& derLength, CW_SignResult& result) {
00459     bool ret = false;
00460     uint8_t apdu[] = { 0x80U, 0xC0U, request.keyType, request.signatureType };
00461
00462 #if CW_DEBUG_LOGGING
00463     _logger.println(F("Sending SIGN APDU..."));
00464 #endif
00465
00466     if (_secure.aesCbcEncrypt(request.session, apdu, sizeof(apdu), data, dataLength,
00467                               derResponse, &derLength)) {
00468         ret = true;
00469     }
00470     else {
00471 #if CW_DEBUG_LOGGING
00472         _logger.println(F("Sign APDU failed."));
00473 #endif
00474         result.errorCode = CW_SIGN_NO_KEY_LOADED;
00475     }
00476
00477     return ret;
00478 }
00479
00480 bool CryptnoxWallet::extractRawSignature(const uint8_t* derResponse, uint16_t derLength,
00481                                          CW_SignResult& result) {
00482     bool ret = false;
00483
00484     if ((derLength < 2U) || (derResponse[0] != CW_DER_TAG_SEQUENCE)) {
00485 #if CW_DEBUG_LOGGING
00486         _logger.println(F("Error: Invalid signature data (missing DER SEQUENCE tag)."));
00487 #endif
00488         result.errorCode = CW_NOK;
00489     }
00490     else {
00491         uint8_t derContentLength = derResponse[1];
00492         uint8_t derTotalLength   = 2U + derContentLength;
00493
00494         if (derTotalLength > derLength) {
00495 #if CW_DEBUG_LOGGING
00496             _logger.println(F("Error: DER signature length exceeds response."));
00497 #endif
00498             result.errorCode = CW_NOK;
00499         }
00500         else {
00501             uint8_t r[33U] = { 0U };
00502             uint8_t s[33U] = { 0U };
00503             uint8_t rLen = 0U;
00504             uint8_t sLen = 0U;
00505
00506             if (!parseDerSignature(derResponse, derTotalLength, r, rLen, s, sLen)) {
00507 #if CW_DEBUG_LOGGING
00508                 _logger.println(F("Error: Failed to parse DER signature."));
00509 #endif
```

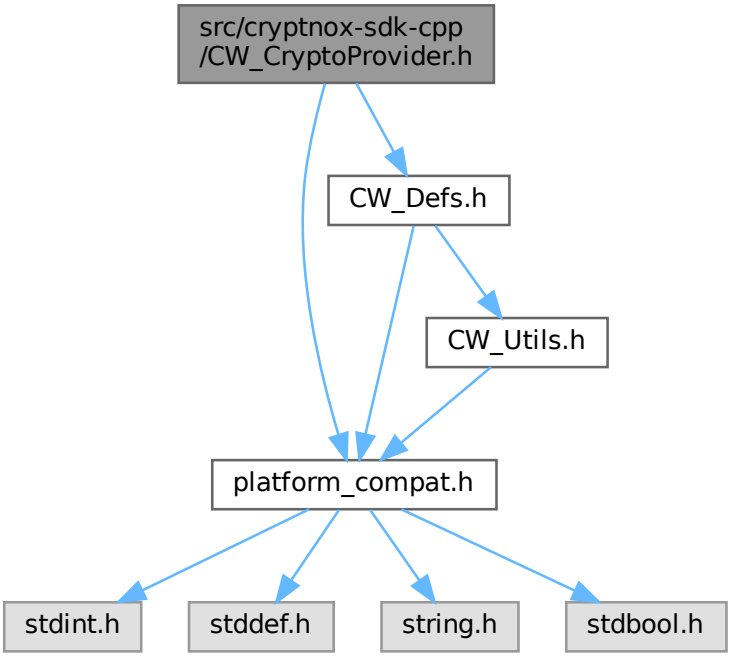
```
00510                 result.errorCode = CW_NOK;
00511             }
00512             else {
00513                 memset(result.signature, 0U, CW_RAW_SIGNATURE_SIZE);
00514 
00515                 if (rLen > 0U) {
00516                     uint8_t rSrc = 0U;
00517                     uint8_t rDstLen = 32U;
00518                     if ((rLen == 33U) && (r[0] == 0x00U)) { rSrc = 1U; rLen = 32U; }
00519                     if (rLen <= rDstLen) {
00520                         (void)CW_Utils::safe_memcpy(result.signature + (rDstLen - rLen),
      static_cast<size_t>(rDstLen + rLen), r + rSrc, rLen);
00521                     }
00522                 }
00523 
00524                 if (sLen > 0U) {
00525                     uint8_t sSrc = 0U;
00526                     uint8_t sDstLen = 32U;
00527                     if ((sLen == 33U) && (s[0] == 0x00U)) { sSrc = 1U; sLen = 32U; }
00528                     if (sLen <= sDstLen) {
00529                         (void)CW_Utils::safe_memcpy(result.signature + 32U + (sDstLen - sLen),
      static_cast<size_t>(sLen), s + sSrc, sLen);
00530                     }
00531                 }
00532 
00533                 ret = true;
00534             }
00535             CW_Utils::secure_wipe(r, sizeof(r));
00536             CW_Utils::secure_wipe(s, sizeof(s));
00537         }
00538     }
00539 
00540     return ret;
00541 }
00542 
00543 void CryptnoxWallet::debugPrintSignature(const uint8_t* signature) {
00544 #if CW_DEBUG_LOGGING
00545     _logger.print(F("Signature ("));
00546     _logger.print((uint8_t)CW_RAW_SIGNATURE_SIZE);
00547     _logger.println(F(" bytes):"));
00548     for (uint8_t i = 0U; i < CW_RAW_SIGNATURE_SIZE; i++) {
00549         _logger.print(F("0x"));
00550         if (signature[i] < 0x10U) { _logger.print(F("0")); }
00551         _logger.print(signature[i], HEX);
00552         _logger.print(F(" "));
00553         if (((i + 1U) % 16U == 0U) && ((i + 1U) != CW_RAW_SIGNATURE_SIZE)) { _logger.println(); }
00554     }
00555     _logger.println();
00556 #else
00557     (void)signature;
00558 #endif
00559 }
```

6.35 src/cryptnox-sdk-cpp/CW_CryptoProvider.h File Reference

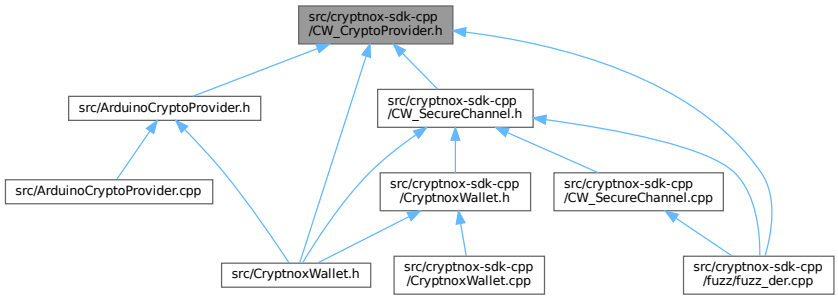
Abstract cryptographic primitives interface.

```
#include "platform_compat.h"
#include "CW_Defs.h"
```

Include dependency graph for CW_CryptoProvider.h:



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



Classes

- class CW_CryptoProvider

  Abstract interface for cryptographic operations used by CW_SecureChannel.

6.35.1 Detailed Description

Abstract cryptographic primitives interface.
Declares CW_CryptoProvider, the contract that any concrete cryptographic backend (mbedTLS, BearSSL, OpenSSL, ESP32 hardware crypto, micro-ecc + AESLib + SHA512, ...) must implement so the secure channel remains decoupled from any specific crypto library.

Provides: SHA-256/512, AES-CBC encrypt/decrypt, ECDH shared secret, EC key pair generation, and cryptographically secure RNG.
One of the three adapter interfaces a host integration must provide.
Definition in file CW_CryptoProvider.h.

6.36 CW_CryptoProvider.h

Go to the documentation of this file.

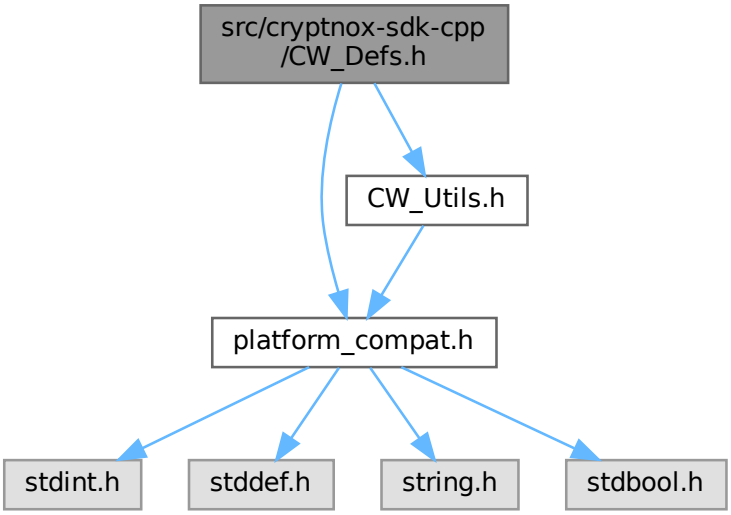
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00020
00021 #ifndef CW_CRYPTOPROVIDER_H
00022 #define CW_CRYPTOPROVIDER_H
00023
00024 /**********************************************************************
00025  * 1. Included files
00026  **********************************************************************/
00027
00028 #include "platform_compat.h"
00029 #include "CW_Defs.h"
00030
00031 /**********************************************************************
00032  * 2. Class declaration
00033  **********************************************************************/
00034
00045 class CW_CryptoProvider {
00046 public:
00055     virtual bool sha256(const uint8_t* data, size_t len, uint8_t* out) = 0;
00056
00065     virtual bool sha512(const uint8_t* data, size_t len, uint8_t* out) = 0;
00066
00080     virtual uint16_t aesCbcEncrypt(const uint8_t* in, uint16_t len, uint8_t* out,
00081                                    const uint8_t* key, uint8_t keyLen,
00082                                    uint8_t* iv, bool bitPadding) = 0;
00083
00096     virtual uint16_t aesCbcDecrypt(uint8_t* in, uint16_t len, uint8_t* out,
00097                                    const uint8_t* key, uint8_t keyLen,
00098                                    uint8_t* iv, bool bitPadding) = 0;
00099
00109     virtual bool ecdh(const uint8_t* pubKey, const uint8_t* privKey,
00110                       uint8_t* secret, CW_Curve curve) = 0;
00111
00120     virtual bool makeKey(uint8_t* pubKey, uint8_t* privKey,
00121                          CW_Curve curve) = 0;
00122
00130     virtual bool random(uint8_t* dest, unsigned size) = 0;
00131
00142     virtual bool ecdsaVerify(const uint8_t* pubKey64,
00143                              const uint8_t* hash, size_t hashLen,
00144                              const uint8_t* sig, CW_Curve curve) = 0;
00145
00146     virtual ~CW_CryptoProvider() {}
00147 };
00148
00149 #endif // CW_CRYPTOPROVIDER_H
```

6.37 src/cryptnox-sdk-cpp/CW_Defs.h File Reference

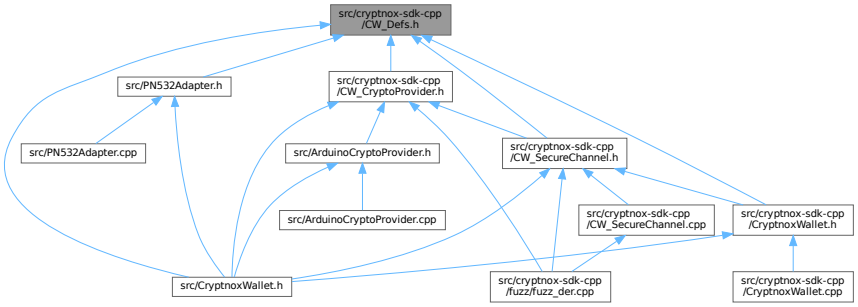
Shared constants, error codes, and session state for the SDK.

```
#include "platform_compat.h"
#include "CW_Utils.h"
```

Include dependency graph for CW_Defs.h:



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



Classes

- struct CW_SecureSession

  Holds cryptographic session state for reentrant secure channel operations.

Macros

- #define CW_AESKEY_SIZE (32U)
- #define CW_MACKEY_SIZE (32U)
- #define CW_IV_SIZE (16U)
- #define CW_OK (0x00U)
- #define CW_NOK (0x01U)
- #define CW_INVALID_SESSION (0x02U)
- #define CW_SIGN_CURR_K1 (0x00U)
- #define CW_SIGN_CURR_R1 (0x10U)

- #define CW_SIGN_DERIVE_K1 (0x01U)
- #define CW_SIGN_DERIVE_R1 (0x11U)
- #define CW_SIGN_PINLESS_K1 (0x03U)
- #define CW_SIGN_WITH_PIN (false)
- #define CW_SIGN_PINLESS (true)
- #define CW_SIGN_SIG_ECDSA_LOW_S (0x00U)
- #define CW_SIGN_SIG_ECDSA_EOSIO (0x01U)
- #define CW_SIGN_SIG_SCHNORR_BIP340 (0x02U)
- #define CW_SIGN_KEY_TOO_SHORT (0x80U)
- #define CW_SIGN_NO_KEY_LOADED (0x81U)
- #define CW_SIGN_PIN_INCORRECT (0x82U)
- #define CW_SIGN_KEY_TOO_SHORT_WITH_PINLESS_MODE (0x83U)
- #define CW_RAW_SIGNATURE_SIZE (64U)
- #define CW_HASH_SIZE (32U)
- #define CW_MAX_DERIVE_PATH_LENGTH (20U)
- #define CW_MIN_PIN_LENGTH (4U)
- #define CW_MAX_PIN_LENGTH (9U)
- #define CW_USER_DATA_PAGE_SIZE (208U)
- #define CW_CONNECT_MAX_ATTEMPTS (5U)
- #define CW_SIG_R_OFFSET (0U)
- #define CW_SIG_S_OFFSET (32U)
- #define CW_DER_TAG_SEQUENCE (0x30U)
- #define CW_DER_TAG_INTEGER (0x02U)
- #define CW_CERT_NONCE_SIZE (8U)
- #define CW_CERT_OK (0x00U)
- #define CW_CERT_FORMAT_ERROR (0x10U)
- #define CW_CERT_NONCE_MISMATCH (0x11U)
- #define CW_CERT_CARD_SIG_INVALID (0x12U)
- #define CW_CERT_MANUF_SIG_INVALID (0x13U)
- #define CW_CERT_KEY_NOT_FOUND (0x14U)
- #define CW_MANUF_CERT_MAX_BYTES (420U)
- #define CW_VERIFY_CERT 1
- #define CW_DEBUG_LOGGING 0

Enumerations

- enum CW_Curve { CW_CURVE_SECP256R1 = 0 , CW_CURVE_SECP256K1 = 1 }
  Portable curve identifier used throughout the SDK.

6.37.1 Detailed Description

Shared constants, error codes, and session state for the SDK.
Defines:

- AES session key / IV sizes
- Generic and SIGN-specific error codes (`CW_OK`, `CW_NOK`, `CW_SIGN_*`)
- SIGN APDU parameter values (key types, signature types, PIN/PIN-less)
- Buffer and protocol size limits
- Certificate verification result codes (`CW_CERT_*`)
- CW_Curve enum (portable curve identifier)
- CW_SecureSession (encryption + MAC keys + rolling IV)
- Compile-time security gates: `CW_VERIFY_CERT`, `CW_DEBUG_LOGGING`

Definition in file CW_Defs.h.

6.37.2 Macro Definition Documentation

6.37.2.1 CW_AESKEY_SIZE

```
#define CW_AESKEY_SIZE (32U)
```

AES-256 session encryption key size in bytes
Definition at line 75 of file CW_Defs.h.
Referenced by CryptnoxWallet::isSecureChannelOpen(), and CW_SecureChannel::mutuallyAuthenticate().

6.37.2.2 CW_CERT_CARD_SIG_INVALID

```
#define CW_CERT_CARD_SIG_INVALID (0x12U)
```

Card cert ECDSA sig failed
Definition at line 130 of file CW_Defs.h.
Referenced by CW_SecureChannel::verifyCertificateChain().

6.37.2.3 CW_CERT_FORMAT_ERROR

```
#define CW_CERT_FORMAT_ERROR (0x10U)
```

Malformed certificate data
Definition at line 128 of file CW_Defs.h.
Referenced by CW_SecureChannel::verifyCertificateChain().

6.37.2.4 CW_CERT_KEY_NOT_FOUND

```
#define CW_CERT_KEY_NOT_FOUND (0x14U)
```

Device public key OID not found
Definition at line 132 of file CW_Defs.h.
Referenced by CW_SecureChannel::verifyCertificateChain().

6.37.2.5 CW_CERT_MANUF_SIG_INVALID

```
#define CW_CERT_MANUF_SIG_INVALID (0x13U)
```

Manufacturer cert ECDSA sig failed
Definition at line 131 of file CW_Defs.h.
Referenced by CW_SecureChannel::verifyCertificateChain().

6.37.2.6 CW_CERT_NONCE_MISMATCH

```
#define CW_CERT_NONCE_MISMATCH (0x11U)
```

Challenge nonce not echoed
Definition at line 129 of file CW_Defs.h.
Referenced by CW_SecureChannel::verifyCertificateChain().

6.37.2.7 CW_CERT_NONCE_SIZE

```
#define CW_CERT_NONCE_SIZE (8U)
```

Challenge nonce length in bytes
Definition at line 124 of file CW_Defs.h.
Referenced by CW_SecureChannel::verifyCertificateChain().

6.37.2.8 CW_CERT_OK

```
#define CW_CERT_OK (0x00U)
```

Certificate chain verified
Definition at line 127 of file CW_Defs.h.
Referenced by CryptnoxWallet::establishSecureChannel(), and CW_SecureChannel::verifyCertificateChain().

6.37.2.9 CW_CONNECT_MAX_ATTEMPTS

```
#define CW_CONNECT_MAX_ATTEMPTS (5U)
```

Max NFC connection retry attempts
Definition at line 113 of file CW_Defs.h.
Referenced by CryptnoxWallet::connect().

6.37.2.10 CW_DEBUG_LOGGING

```
#define CW_DEBUG_LOGGING 0
```

Set to 1 to enable library-internal debug logging via CW_Logger.
Off by default. Enabling it kills flash optimisation — measured on Arduino UNO R4 (Renesas RA4↩ M1): +149 KB flash, +6 KB SRAM (31 % → 88 % of a 256 KB image on the Sign example). Same order of magnitude on every constrained MCU.
Also leaks session state over UART (SEC-012). Bring-up only, never in release builds.
Definition at line 215 of file CW_Defs.h.

6.37.2.11 CW_DER_TAG_INTEGER

```
#define CW_DER_TAG_INTEGER (0x02U)
```

Definition at line 121 of file CW_Defs.h.
Referenced by CryptnoxWallet::parseDerSignature().

6.37.2.12 CW_DER_TAG_SEQUENCE

```
#define CW_DER_TAG_SEQUENCE (0x30U)
```

Definition at line 120 of file CW_Defs.h.
Referenced by CryptnoxWallet::extractRawSignature(), and CryptnoxWallet::parseDerSignature().

6.37.2.13 CW_HASH_SIZE

```
#define CW_HASH_SIZE (32U)
```

Standard hash size

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 108 of file CW_Defs.h.
Referenced by CryptnoxWallet::buildSignPayload(), loop(), setup(), CryptnoxWallet::sign(), and CryptnoxWallet::validateSignRequest().

6.37.2.14 CW_INVALID_SESSION

```
#define CW_INVALID_SESSION (0x02U)
```

Invalid session
Definition at line 82 of file CW_Defs.h.
Referenced by CryptnoxWallet::validateSignRequest().

6.37.2.15 CW_IV_SIZE

```
#define CW_IV_SIZE (16U)
```

AES-CBC IV size in bytes
Definition at line 77 of file CW_Defs.h.
Referenced by CW_SecureChannel::aesCbcEncrypt(), and CW_SecureChannel::mutuallyAuthenticate().

6.37.2.16 CW_MACKEY_SIZE

```
#define CW_MACKEY_SIZE (32U)
```

AES-256 session MAC key size in bytes
Definition at line 76 of file CW_Defs.h.
Referenced by CW_SecureChannel::mutuallyAuthenticate().

6.37.2.17 CW_MANUF_CERT_MAX_BYTES

```
#define CW_MANUF_CERT_MAX_BYTES (420U)
```

Definition at line 136 of file CW_Defs.h.
Referenced by CW_SecureChannel::getManufacturerCertificate().

6.37.2.18 CW_MAX_DERIVE_PATH_LENGTH

```
#define CW_MAX_DERIVE_PATH_LENGTH (20U)
```

Max BIP32 path bytes
Definition at line 109 of file CW_Defs.h.
Referenced by CryptnoxWallet::buildSignPayload(), and CryptnoxWallet::sign().

6.37.2.19 CW_MAX_PIN_LENGTH

```
#define CW_MAX_PIN_LENGTH (9U)
```

Maximum PIN length
Definition at line 111 of file CW_Defs.h.
Referenced by CryptnoxWallet::buildSignPayload(), CryptnoxWallet::sign(), CryptnoxWallet::validateSignRequest(), and CryptnoxWallet::verifyPin().

6.37.2.20 CW_MIN_PIN_LENGTH

```
#define CW_MIN_PIN_LENGTH (4U)
```

Minimum PIN length
Definition at line 110 of file CW_Defs.h.
Referenced by CryptnoxWallet::validateSignRequest(), and CryptnoxWallet::verifyPin().

6.37.2.21 CW_NOK

```
#define CW_NOK (0x01U)
```

NOK
Definition at line 81 of file CW_Defs.h.
Referenced by CW_SignResult::CW_SignResult(), and CryptnoxWallet::extractRawSignature().

6.37.2.22 CW_OK

```
#define CW_OK (0x00U)
```

OK

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 80 of file CW_Defs.h.
Referenced by loop(), setup(), and CryptnoxWallet::sign().

6.37.2.23 CW_RAW_SIGNATURE_SIZE

```
#define CW_RAW_SIGNATURE_SIZE (64U)
```

Raw signature (r[32] + s[32])
Definition at line 107 of file CW_Defs.h.
Referenced by CryptnoxWallet::debugPrintSignature(), and CryptnoxWallet::extractRawSignature().

6.37.2.24 CW_SIG_R_OFFSET

```
#define CW_SIG_R_OFFSET (0U)
```
Byte offset of the r component

Examples

BasicUsage.ino, and Sign.ino.

Definition at line 116 of file CW_Defs.h.
Referenced by loop().

6.37.2.25 CW_SIG_S_OFFSET

```
#define CW_SIG_S_OFFSET (32U)
```
Byte offset of the s component

Examples

BasicUsage.ino, and Sign.ino.

Definition at line 117 of file CW_Defs.h.
Referenced by loop().

6.37.2.26 CW_SIGN_CURR_K1

```
#define CW_SIGN_CURR_K1 (0x00U)
```
Current key (k1)

Examples

BasicUsage.ino, and Sign.ino.

Definition at line 85 of file CW_Defs.h.
Referenced by CW_SignRequest::CW_SignRequest(), and loop().

6.37.2.27 CW_SIGN_CURR_R1

```
#define CW_SIGN_CURR_R1 (0x10U)
```
Current key (r1)
Definition at line 86 of file CW_Defs.h.

6.37.2.28 CW_SIGN_DERIVE_K1

```
#define CW_SIGN_DERIVE_K1 (0x01U)
```
Derive with k1 curve

Examples

UsdcSigning.ino.

Definition at line 87 of file CW_Defs.h.
Referenced by CryptnoxWallet::buildSignPayload(), and setup().

6.37.2.29 CW_SIGN_DERIVE_R1

```
#define CW_SIGN_DERIVE_R1 (0x11U)
```
Derive with r1 curve
Definition at line 88 of file CW_Defs.h.
Referenced by CryptnoxWallet::buildSignPayload().

6.37.2.30 CW_SIGN_KEY_TOO_SHORT

```
#define CW_SIGN_KEY_TOO_SHORT (0x80U)
```

Definition at line 101 of file CW_Defs.h.
Referenced by CryptnoxWallet::validateSignRequest().

6.37.2.31 CW_SIGN_KEY_TOO_SHORT_WITH_PINLESS_MODE

```
#define CW_SIGN_KEY_TOO_SHORT_WITH_PINLESS_MODE (0x83U)
```

Definition at line 104 of file CW_Defs.h.
Referenced by CryptnoxWallet::validateSignRequest().

6.37.2.32 CW_SIGN_NO_KEY_LOADED

```
#define CW_SIGN_NO_KEY_LOADED (0x81U)
```

Examples

UsdcSigning.ino.

Definition at line 102 of file CW_Defs.h.
Referenced by CryptnoxWallet::sendSignApdu(), and setup().

6.37.2.33 CW_SIGN_PIN_INCORRECT

```
#define CW_SIGN_PIN_INCORRECT (0x82U)
```

Examples

Sign.ino, and UsdcSigning.ino.

Definition at line 103 of file CW_Defs.h.
Referenced by loop(), setup(), and CryptnoxWallet::validateSignRequest().

6.37.2.34 CW_SIGN_PINLESS

```
#define CW_SIGN_PINLESS (true)
```

PIN-less path
Definition at line 93 of file CW_Defs.h.

6.37.2.35 CW_SIGN_PINLESS_K1

```
#define CW_SIGN_PINLESS_K1 (0x03U)
```

PIN-less path (k1 only)
Definition at line 89 of file CW_Defs.h.
Referenced by CryptnoxWallet::validateSignRequest().

6.37.2.36 CW_SIGN_SIG_ECDSA_EOSIO

```
#define CW_SIGN_SIG_ECDSA_EOSIO (0x01U)
```

ECDSA EOSIO format
Definition at line 97 of file CW_Defs.h.

6.37.2.37 CW_SIGN_SIG_ECDSA_LOW_S

```
#define CW_SIGN_SIG_ECDSA_LOW_S (0x00U)
```

ECDSA with canonical low S

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 96 of file CW_Defs.h.
Referenced by CW_SignRequest::CW_SignRequest(), loop(), and setup().

6.37.2.38 CW_SIGN_SIG_SCHNORR_BIP340

```
#define CW_SIGN_SIG_SCHNORR_BIP340 (0x02U)
```

Schnorr BIP340
Definition at line 98 of file CW_Defs.h.

6.37.2.39 CW_SIGN_WITH_PIN

```
#define CW_SIGN_WITH_PIN (false)
```

PIN path

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 92 of file CW_Defs.h.
Referenced by CW_SignRequest::CW_SignRequest(), loop(), and setup().

6.37.2.40 CW_USER_DATA_PAGE_SIZE

```
#define CW_USER_DATA_PAGE_SIZE (208U)
```

Max plaintext bytes per write user data page
Definition at line 112 of file CW_Defs.h.
Referenced by CryptnoxWallet::writeUserData().

6.37.2.41 CW_VERIFY_CERT

```
#define CW_VERIFY_CERT 1
```

Certificate chain verification is always enabled (SEC-004 / H-07). Building with -DCW_VERIFY_↩ CERT=0 is a hard error — it disables the card authenticity gate and allows any forged key to be accepted.
Definition at line 196 of file CW_Defs.h.

6.38 CW_Defs.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
00020
00021 #ifndef CW_DEFS_H
00022 #define CW_DEFS_H
00023
00024 /**************************************************************
00025  * 0. Doxygen module groups (used by @ingroup throughout the SDK)
00026  **************************************************************/
00027
00035
00044
00053
00062
00063 /**************************************************************
00064  * 1. Included files
00065  **************************************************************/
00066
00067 #include "platform_compat.h"
00068 #include "CW_Utils.h"
00069
00070 /**************************************************************
00071  * 2. Constants / define declarations
00072  **************************************************************/
00073
00074 /* Session key sizes */
00075 #define CW_AESKEY_SIZE    (32U)
00076 #define CW_MACKEY_SIZE    (32U)
00077 #define CW_IV_SIZE        (16U)
00078
00079 /* Generic error codes */
```

```
00080 #define CW_OK                          (0x00U)
00081 #define CW_NOK                         (0x01U)
00082 #define CW_INVALID_SESSION             (0x02U)
00083
00084 /* Key / path types for SIGN command (keyType) */
00085 #define CW_SIGN_CURR_K1                (0x00U)
00086 #define CW_SIGN_CURR_R1                (0x10U)
00087 #define CW_SIGN_DERIVE_K1              (0x01U)
00088 #define CW_SIGN_DERIVE_R1              (0x11U)
00089 #define CW_SIGN_PINLESS_K1             (0x03U)
00090
00091 /* PIN mode for SIGN command */
00092 #define CW_SIGN_WITH_PIN               (false)
00093 #define CW_SIGN_PINLESS                (true)
00094
00095 /* Signature types for SIGN command */
00096 #define CW_SIGN_SIG_ECDSA_LOW_S        (0x00U)
00097 #define CW_SIGN_SIG_ECDSA_EOSIO        (0x01U)
00098 #define CW_SIGN_SIG_SCHNORR_BIP340     (0x02U)
00099
00100 /* SIGN-specific error codes */
00101 #define CW_SIGN_KEY_TOO_SHORT                  (0x80U)
00102 #define CW_SIGN_NO_KEY_LOADED                  (0x81U)
00103 #define CW_SIGN_PIN_INCORRECT                  (0x82U)
00104 #define CW_SIGN_KEY_TOO_SHORT_WITH_PINLESS_MODE (0x83U)
00105
00106 /* Size constants */
00107 #define CW_RAW_SIGNATURE_SIZE          (64U)
00108 #define CW_HASH_SIZE                   (32U)
00109 #define CW_MAX_DERIVE_PATH_LENGTH      (20U)
00110 #define CW_MIN_PIN_LENGTH               (4U)
00111 #define CW_MAX_PIN_LENGTH               (9U)
00112 #define CW_USER_DATA_PAGE_SIZE         (208U)
00113 #define CW_CONNECT_MAX_ATTEMPTS         (5U)
00114
00115 /* Byte offsets within a raw 64-byte signature (r[32] || s[32]) */
00116 #define CW_SIG_R_OFFSET                (0U)
00117 #define CW_SIG_S_OFFSET                (32U)
00118
00119 /* DER encoding tags (ASN.1) */
00120 #define CW_DER_TAG_SEQUENCE            (0x30U)
00121 #define CW_DER_TAG_INTEGER             (0x02U)
00122
00123 /* Certificate verification constants */
00124 #define CW_CERT_NONCE_SIZE             (8U)
00125
00126 /* Certificate verification result codes */
00127 #define CW_CERT_OK                     (0x00U)
00128 #define CW_CERT_FORMAT_ERROR           (0x10U)
00129 #define CW_CERT_NONCE_MISMATCH         (0x11U)
00130 #define CW_CERT_CARD_SIG_INVALID       (0x12U)
00131 #define CW_CERT_MANUF_SIG_INVALID      (0x13U)
00132 #define CW_CERT_KEY_NOT_FOUND          (0x14U)
00133
00134 /* Manufacturer certificate maximum buffer size (bytes).
00135  * Actual Cryptnox Basic G1 manufacturer certificate is 411 bytes (0x019B). */
00136 #define CW_MANUF_CERT_MAX_BYTES        (420U)
00137
00138 /****************************************************************
00139  * 3. CW_Curve enum
00140  ****************************************************************/
00141
00151 enum CW_Curve {
00152     CW_CURVE_SECP256R1 = 0,
00153     CW_CURVE_SECP256K1 = 1
00154 };
00155
00156 /****************************************************************
00157  * 4. CW_SecureSession struct
00158  ****************************************************************/
00159
00168 struct CW_SecureSession {
00169     uint8_t aesKey[CW_AESKEY_SIZE];
00170     uint8_t macKey[CW_MACKEY_SIZE];
00171     uint8_t iv[CW_IV_SIZE];
00172
00174     CW_SecureSession() {
00175         memset(aesKey, 0U, sizeof(aesKey));
00176         memset(macKey, 0U, sizeof(macKey));
00177         memset(iv, 0U, sizeof(iv));
00178     }
00179
00181     void clear() {
00182         CW_Utils::secure_wipe(aesKey, sizeof(aesKey));
00183         CW_Utils::secure_wipe(macKey, sizeof(macKey));
00184         CW_Utils::secure_wipe(iv,     sizeof(iv));
00185     }
```

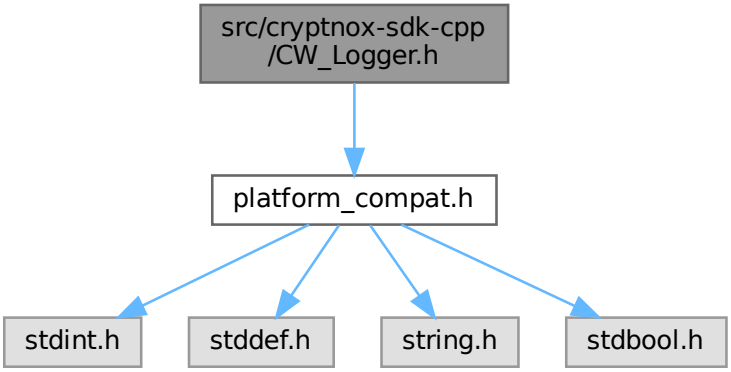
```
00186 };
00187
00188 /**************************************************************
00189  * 5. Compile-time feature flags
00190  **************************************************************/
00191
00195 #ifndef CW_VERIFY_CERT
00196 #define CW_VERIFY_CERT 1
00197 #endif
00198 #if CW_VERIFY_CERT == 0
00199 #  error "CW_VERIFY_CERT=0 disables certificate chain verification (CRIT-02/H-07). " \
00200         "Remove -DCW_VERIFY_CERT=0 from your build flags -- this gate must never be disabled."
00201 #endif
00202
00214 #ifndef CW_DEBUG_LOGGING
00215 #  define CW_DEBUG_LOGGING 0
00216 #endif
00217
00218 #if CW_DEBUG_LOGGING && defined(NDEBUG)
00219 #  error "CW_DEBUG_LOGGING=1 is set but NDEBUG is defined (release/optimised build). " \
00220         "Debug logging must not ship in production firmware -- it leaks session state " \
00221         "over UART.  Remove -DCW_DEBUG_LOGGING=1 from your release build flags (SEC-012)."
00222 #endif
00223
00224 #endif // CW_DEFS_H
```

6.39 src/cryptnox-sdk-cpp/CW_Logger.h File Reference

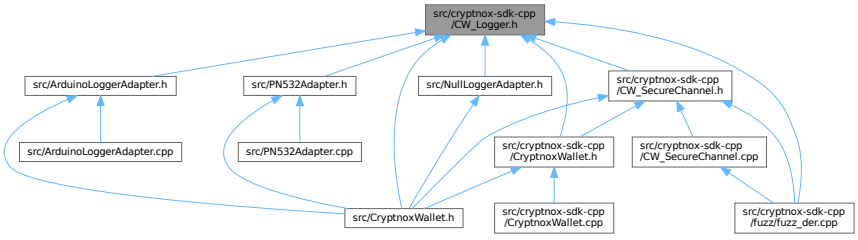
Abstract logging interface.

`#include "platform_compat.h"`

Include dependency graph for CW_Logger.h:



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



Classes

- class CW_Logger

  Abstract interface for serial/debug output.

6.39.1 Detailed Description

Abstract logging interface.
Declares CW_Logger, the contract that any concrete output sink (UART, USB CDC, stdout, syslog, network, ...) must implement so the SDK remains independent of the host platform's logging facility.
The Arduino-style print/println overloads keep `F("...")`-quoted string literals in flash on Arduino targets; on non-Arduino targets `F()` is the identity macro (see platform_compat.h).
One of the three adapter interfaces a host integration must provide.
Definition in file CW_Logger.h.

6.40 CW_Logger.h

Go to the documentation of this file.

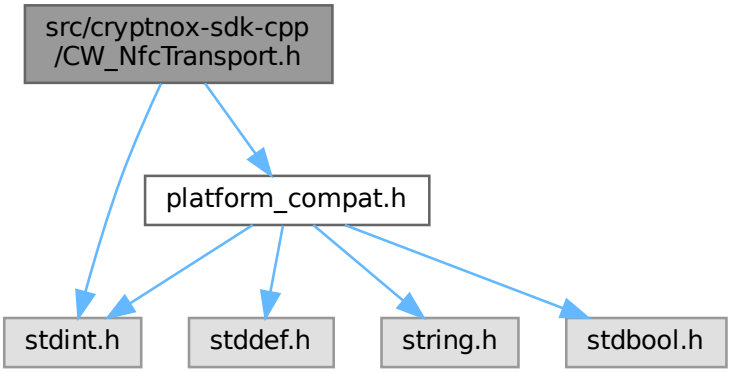
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00020
00021 #ifndef CW_LOGGER_H
00022 #define CW_LOGGER_H
00023
00024 /**************************************************************************
00025  * 1. Included files
00026  *************************************************************************/
00027
00028 #include "platform_compat.h"
00029
00030 /**************************************************************************
00031  * 2. Class declaration
00032  *************************************************************************/
00033
00048 class CW_Logger {
00049 public:
00055     virtual bool begin(unsigned long baudRate = 115200UL) = 0;
00056
00059     virtual void print(const __FlashStringHelper* str) = 0;
00060     virtual void print(const char* str) = 0;
00061     virtual void print(char c) = 0;
00062     virtual void print(uint8_t value, int base = DEC) = 0;
00063     virtual void print(uint16_t value, int base = DEC) = 0;
00064     virtual void print(uint32_t value, int base = DEC) = 0;
00065     virtual void print(int value, int base = DEC) = 0;
00067
00070     virtual void println() = 0;
00071     virtual void println(const __FlashStringHelper* str) = 0;
00072     virtual void println(const char* str) = 0;
00073     virtual void println(char c) = 0;
00074     virtual void println(uint8_t value, int base = DEC) = 0;
00075     virtual void println(uint16_t value, int base = DEC) = 0;
00076     virtual void println(uint32_t value, int base = DEC) = 0;
00077     virtual void println(int value, int base = DEC) = 0;
00079
00080     virtual ~CW_Logger() {}
00081 };
00082
00083 #endif // CW_LOGGER_H
```

6.41 src/cryptnox-sdk-cpp/CW_NfcTransport.h File Reference

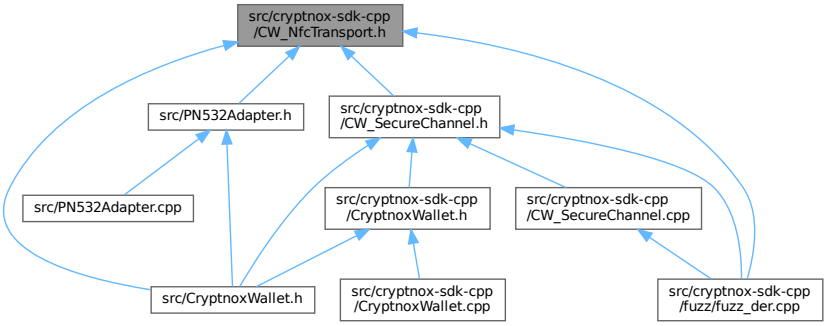
Abstract NFC transport interface.

```
#include "platform_compat.h"
#include <stdint.h>
```

Include dependency graph for CW_NfcTransport.h:



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



Classes

- class CW_NfcTransport

  Abstract interface for NFC transport operations.

6.41.1 Detailed Description

Abstract NFC transport interface.
Declares CW_NfcTransport, the contract that any concrete NFC reader driver (PN532, PN7150, PC/↩ SC, . . .) must implement so that CW_SecureChannel and CryptnoxWallet remain hardware-agnostic.
One of the three adapter interfaces a host integration must provide.
Definition in file CW_NfcTransport.h.

6.42 CW_NfcTransport.h

Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
```

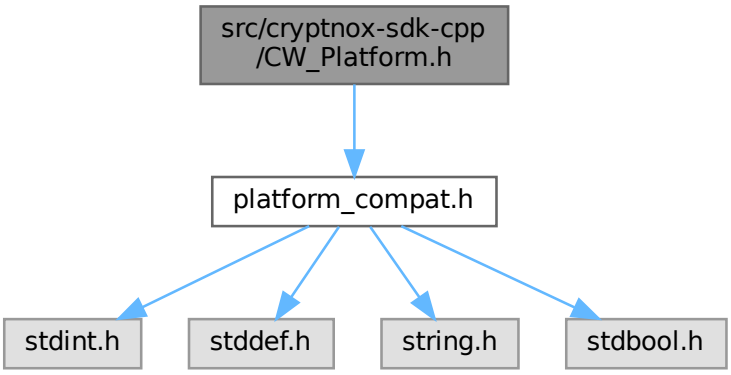
```
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00016
00017 #ifndef CW_NFCTRANSPORT_H
00018 #define CW_NFCTRANSPORT_H
00019
00020 /**************************************************************************
00021  * 1. Included files
00022  *************************************************************************/
00023
00024 #include "platform_compat.h"
00025 #include <stdint.h>
00026
00027 /**************************************************************************
00028  * 2. Class declaration
00029  *************************************************************************/
00030
00040 class CW_NfcTransport {
00041 public:
00046     virtual bool begin() = 0;
00047
00052     virtual bool inListPassiveTarget() = 0;
00053
00063     virtual bool sendAPDU(const uint8_t* apdu, uint8_t apduLen,
00064                           uint8_t* response, uint8_t& responseLen) = 0;
00065
00081     virtual bool sendAPDULarge(const uint8_t* apdu, uint8_t apduLen,
00082                                uint8_t* response, uint16_t& responseLen) {
00083         uint8_t smallLen = static_cast<uint8_t>(
00084             (responseLen > static_cast<uint16_t>(UINT8_MAX))
00085                 ? static_cast<uint16_t>(UINT8_MAX)
00086                 : responseLen);
00087         bool result = sendAPDU(apdu, apduLen, response, smallLen);
00088         responseLen = static_cast<uint16_t>(smallLen);
00089         return result;
00090     }
00091
00095     virtual void resetReader() = 0;
00096
00101     virtual bool printFirmwareVersion() = 0;
00102
00103     virtual ~CW_NfcTransport() {}
00104 };
00105
00106 #endif // CW_NFCTRANSPORT_H
```

6.43 src/cryptnox-sdk-cpp/CW_Platform.h File Reference

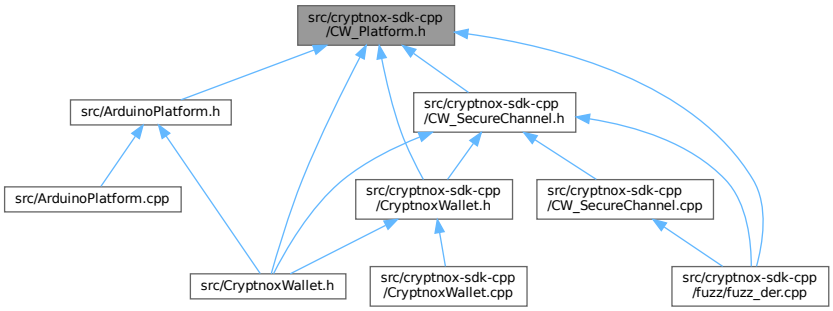
Abstract platform interface for timing primitives.

`#include "platform_compat.h"`

Include dependency graph for CW_Platform.h:



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



Classes

- class CW_Platform

  Abstract interface for platform-specific operations used by the SDK.

6.43.1 Detailed Description

Abstract platform interface for timing primitives.
Declares CW_Platform, the contract that hosts implement so the SDK stays independent of any specific RTOS or bare-metal delay mechanism.
Currently exposes a single operation (CW_Platform::sleep_ms) used by CW_SecureChannel for inter-APDU spacing on slow NFC stacks.
Definition in file CW_Platform.h.

6.44 CW_Platform.h

Go to the documentation of this file.

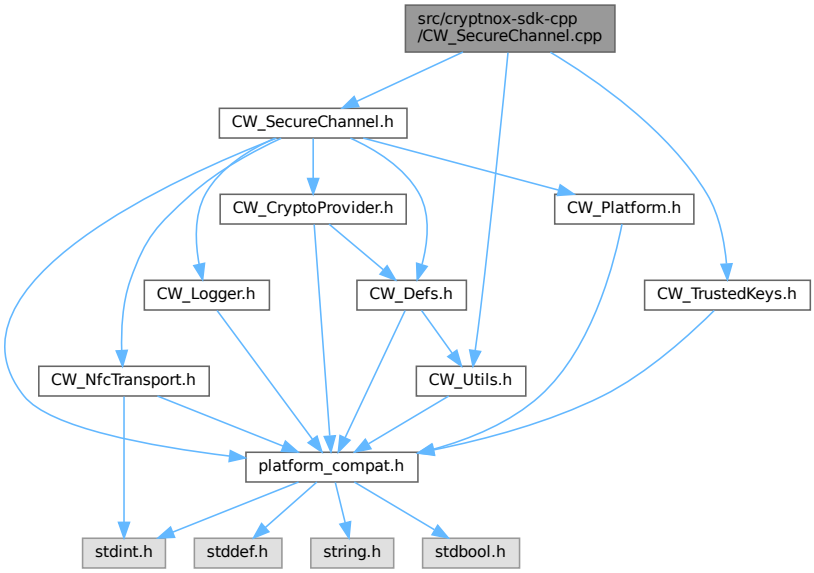
```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00016
00017 #ifndef CW_PLATFORM_H
00018 #define CW_PLATFORM_H
00019
00020 /**********************************************************************
00021  * 1. Included files
00022  **********************************************************************/
00023
00024 #include "platform_compat.h"
00025
00026 /**********************************************************************
00027  * 2. Class declaration
00028  **********************************************************************/
00029
00039 class CW_Platform {
00040 public:
00046     virtual void sleep_ms(uint32_t ms) = 0;
00047
00048     virtual ~CW_Platform() {}
00049 };
00050
00051 #endif /* CW_PLATFORM_H */
```

6.45 src/cryptnox-sdk-cpp/CW_SecureChannel.cpp File Reference

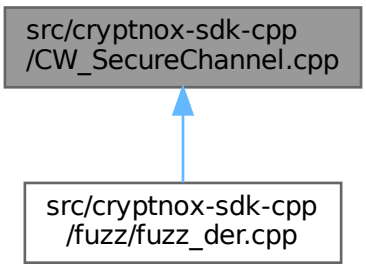
Implementation of the Cryptnox secure channel protocol.

```
#include "CW_SecureChannel.h"
#include "CW_Utils.h"
#include "CW_TrustedKeys.h"
```

Include dependency graph for CW_SecureChannel.cpp:



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



Macros

- #define RESPONSE_GETCARDCERTIFICATE_IN_BYTES 148U
- #define RESPONSE_SELECT_IN_BYTES 40U
- #define RESPONSE_GETMANUFACTURERCERT_PAGE_IN_BYTES 420U
- #define RESPONSE_OPENSECURECHANNEL_IN_BYTES 34U

- #define REQUEST_MUTUALLYAUTHENTICATE_IN_BYTES 69U
- #define RESPONSE_MUTUALLYAUTHENTICATE_IN_BYTES 66U
- #define RESPONSE_STATUS_WORDS_IN_BYTES 2U
- #define OPENSECURECHANNEL_SALT_IN_BYTES (RESPONSE_OPENSECURECHANNEL_IN_BYTES - RESPONSE_STATUS_WORDS_IN_BYTES)
- #define GETCARDCERTIFICATE_IN_BYTES (RESPONSE_GETCARDCERTIFICATE_IN_BYTES - RESPONSE_STATUS_WORDS_IN_BYTES)
- #define RANDOM_BYTES 8U
- #define COMMON_PAIRING_DATA CW_PAIRING_DATA
- #define CLIENT_PRIVATE_KEY_SIZE 32U
- #define CLIENT_PUBLIC_KEY_SIZE 64U
- #define CARDEPHEMERALPUBKEY_SIZE 64U
- #define AES_BLOCK_SIZE 16U
- #define APDU_HEADER_LEN (4U)
- #define APDU_LC_LEN (1U)
- #define MAC_APDU_LEN (12U)
- #define INPUT_BUFFER_LIMIT (CW_USER_DATA_PAGE_SIZE)
- #define ENC_BUF_MAX_LEN (INPUT_BUFFER_LIMIT + AES_BLOCK_SIZE)
- #define MAX_MAC_DATA_LEN (APDU_HEADER_LEN + MAC_APDU_LEN + ENC_BUF_MAX_LEN)
- #define SEND_APDU_MAX_LEN (APDU_HEADER_LEN + APDU_LC_LEN + AES_BLOCK_SIZE + ENC_BUF_MAX_LEN)
- #define DER_TAG_SEQUENCE (0x30U) /* SEQUENCE (universal, constructed) */
- #define DER_TAG_BIT_STRING (0x03U) /* BIT STRING */
- #define DER_TAG_CTX0 (0xA0U) /* [0] EXPLICIT — version in v3 TBSCertificate */
- #define DER_LEN_LONG_FLAG (0x80U) /* set = long-form length */
- #define DER_LEN_LONG_1 (0x81U) /* long form, 1 following byte */
- #define DER_LEN_LONG_2 (0x82U) /* long form, 2 following bytes */
- #define DER_EC_UNCOMPRESSED (0x04U) /* uncompressed point prefix */
- #define DER_EC_POINT_BYTES (65U) /* 0x04 || X[32] || Y[32] */
- #define DER_BIT_UNUSED_ZERO (0x00U) /* BIT STRING unused-bits field must be 0 */

Functions

- static bool derReadLength (const uint8_t *buf, uint16_t bufLen, uint16_t &pos, uint16_t &fieldLen)
- static bool derSkipField (const uint8_t *buf, uint16_t bufLen, uint16_t &pos)
- static bool derWalkMfCert (const uint8_t *buf, uint16_t bufLen, uint16_t &tbsMsgStart, uint16_t &tbsMsgLen, const uint8_t *&pubKey65Ptr, const uint8_t *&sigPtr, uint8_t &sigLen)

Variables

- static uint8_t s_apduBuf [SEND_APDU_MAX_LEN]
- static uint8_t s_macBuf [MAX_MAC_DATA_LEN]
- static uint8_t s_dataBuf [ENC_BUF_MAX_LEN]
- static uint8_t s_mfCertBuf [CW_MANUF_CERT_MAX_BYTES]

6.45.1 Detailed Description

Implementation of the Cryptnox secure channel protocol.
Implements the methods declared in CW_SecureChannel.h: APDU framing, certificate chain verification against the trusted CA keys (CW_TrustedKeys.h), ECDH session key derivation, AES-CBC encrypted messaging with rolling IV, and MAC verification on every response.
Module-level static scratch buffers are reused across calls to keep the stack footprint small; secret material is wiped after use.
Definition in file CW_SecureChannel.cpp.

6.45.2 Macro Definition Documentation

6.45.2.1 AES_BLOCK_SIZE

```
#define AES_BLOCK_SIZE 16U
```

Definition at line 45 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::aesCbcDecrypt(), CW_SecureChannel::aesCbcEncrypt(), and CW_SecureChannel::mutuallyAuthenticate().

6.45.2.2 APDU_HEADER_LEN

```
#define APDU_HEADER_LEN (4U)
```

Definition at line 46 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::mutuallyAuthenticate().

6.45.2.3 APDU_LC_LEN

```
#define APDU_LC_LEN (1U)
```

Definition at line 47 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::aesCbcEncrypt(), and CW_SecureChannel::mutuallyAuthenticate().

6.45.2.4 CARDEPHEMERALPUBKEY_SIZE

```
#define CARDEPHEMERALPUBKEY_SIZE 64U
```

Definition at line 44 of file CW_SecureChannel.cpp.

6.45.2.5 CLIENT_PRIVATE_KEY_SIZE

```
#define CLIENT_PRIVATE_KEY_SIZE 32U
```

Definition at line 42 of file CW_SecureChannel.cpp.

6.45.2.6 CLIENT_PUBLIC_KEY_SIZE

```
#define CLIENT_PUBLIC_KEY_SIZE 64U
```

Definition at line 43 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::openSecureChannel().

6.45.2.7 COMMON_PAIRING_DATA

```
#define COMMON_PAIRING_DATA CW_PAIRING_DATA
```

Definition at line 41 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::mutuallyAuthenticate().

6.45.2.8 DER_BIT_UNUSED_ZERO

```
#define DER_BIT_UNUSED_ZERO (0x00U) /* BIT STRING unused-bits field must be 0 */
```

Definition at line 80 of file CW_SecureChannel.cpp.
Referenced by derWalkMfCert().

6.45.2.9 DER_EC_POINT_BYTES

```
#define DER_EC_POINT_BYTES (65U) /* 0x04 || X[32] || Y[32] */
```

Definition at line 79 of file CW_SecureChannel.cpp.
Referenced by derWalkMfCert().

6.45.2.10 DER_EC_UNCOMPRESSED

```
#define DER_EC_UNCOMPRESSED (0x04U) /* uncompressed point prefix */
```

Definition at line 78 of file CW_SecureChannel.cpp.
Referenced by derWalkMfCert().

6.45.2.11 DER_LEN_LONG_1

```
#define DER_LEN_LONG_1 (0x81U) /* long form, 1 following byte */
```

Definition at line 74 of file CW_SecureChannel.cpp.
Referenced by derReadLength().

6.45.2.12 DER_LEN_LONG_2

```
#define DER_LEN_LONG_2 (0x82U) /* long form, 2 following bytes */
```

Definition at line 75 of file CW_SecureChannel.cpp.
Referenced by derReadLength().

6.45.2.13 DER_LEN_LONG_FLAG

```
#define DER_LEN_LONG_FLAG (0x80U) /* set = long-form length */
```

Definition at line 73 of file CW_SecureChannel.cpp.
Referenced by derReadLength().

6.45.2.14 DER_TAG_BIT_STRING

```
#define DER_TAG_BIT_STRING (0x03U) /* BIT STRING */
```

Definition at line 69 of file CW_SecureChannel.cpp.
Referenced by derWalkMfCert().

6.45.2.15 DER_TAG_CTX0

```
#define DER_TAG_CTX0 (0xA0U) /* [0] EXPLICIT -- version in v3 TBSCertificate */
```

Definition at line 70 of file CW_SecureChannel.cpp.
Referenced by derWalkMfCert().

6.45.2.16 DER_TAG_SEQUENCE

```
#define DER_TAG_SEQUENCE (0x30U) /* SEQUENCE (universal, constructed) */
```

Definition at line 68 of file CW_SecureChannel.cpp.
Referenced by derWalkMfCert().

6.45.2.17 ENC_BUF_MAX_LEN

```
#define ENC_BUF_MAX_LEN (INPUT_BUFFER_LIMIT + AES_BLOCK_SIZE)
```

Definition at line 50 of file CW_SecureChannel.cpp.

6.45.2.18 GETCARDCERTIFICATE_IN_BYTES

```
#define GETCARDCERTIFICATE_IN_BYTES (RESPONSE_GETCARDCERTIFICATE_IN_BYTES - RESPONSE_STATUS_WORDS_IN_BYTES)
```

Definition at line 38 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::getCardCertificate().

6.45.2.19 INPUT_BUFFER_LIMIT

```
#define INPUT_BUFFER_LIMIT (CW_USER_DATA_PAGE_SIZE)
```

Definition at line 49 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::aesCbcEncrypt().

6.45.2.20 MAC_APDU_LEN

```
#define MAC_APDU_LEN (12U)
```
Definition at line 48 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::aesCbcEncrypt().

6.45.2.21 MAX_MAC_DATA_LEN

```
#define MAX_MAC_DATA_LEN (APDU_HEADER_LEN + MAC_APDU_LEN + ENC_BUF_MAX_LEN)
```
Definition at line 51 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::aesCbcEncrypt().

6.45.2.22 OPENSECURECHANNEL_SALT_IN_BYTES

```
#define OPENSECURECHANNEL_SALT_IN_BYTES (RESPONSE_OPENSECURECHANNEL_IN_BYTES - RESPONSE_STATUS_WORDS_IN_BYTES)
```
Definition at line 37 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::openSecureChannel().

6.45.2.23 RANDOM_BYTES

```
#define RANDOM_BYTES 8U
```
Definition at line 40 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::getCardCertificate().

6.45.2.24 REQUEST_MUTUALLYAUTHENTICATE_IN_BYTES

```
#define REQUEST_MUTUALLYAUTHENTICATE_IN_BYTES 69U
```
Definition at line 33 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::mutuallyAuthenticate().

6.45.2.25 RESPONSE_GETCARDCERTIFICATE_IN_BYTES

```
#define RESPONSE_GETCARDCERTIFICATE_IN_BYTES 148U
```
Definition at line 27 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::getCardCertificate().

6.45.2.26 RESPONSE_GETMANUFACTURERCERT_PAGE_IN_BYTES

```
#define RESPONSE_GETMANUFACTURERCERT_PAGE_IN_BYTES 420U
```
Definition at line 31 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::getManufacturerCertificate().

6.45.2.27 RESPONSE_MUTUALLYAUTHENTICATE_IN_BYTES

```
#define RESPONSE_MUTUALLYAUTHENTICATE_IN_BYTES 66U
```
Definition at line 34 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::mutuallyAuthenticate().

6.45.2.28 RESPONSE_OPENSECURECHANNEL_IN_BYTES

```
#define RESPONSE_OPENSECURECHANNEL_IN_BYTES 34U
```
Definition at line 32 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::openSecureChannel().

6.45.2.29 RESPONSE_SELECT_IN_BYTES

```
#define RESPONSE_SELECT_IN_BYTES 40U
```
Definition at line 29 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::selectApdu().

6.45.2.30 RESPONSE_STATUS_WORDS_IN_BYTES

```
#define RESPONSE_STATUS_WORDS_IN_BYTES 2U
```

Definition at line 35 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::getCardCertificate(), and CW_SecureChannel::getManufacturerCertificate().

6.45.2.31 SEND_APDU_MAX_LEN

```
#define SEND_APDU_MAX_LEN (APDU_HEADER_LEN + APDU_LC_LEN + AES_BLOCK_SIZE + ENC_BUF_MAX_LEN)
```

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

6.45.3 Function Documentation

6.45.3.1 derReadLength()

```
bool derReadLength (
            const uint8_t * buf,
            uint16_t bufLen,
            uint16_t & pos,
            uint16_t & fieldLen)  [static]
```

Definition at line 732 of file CW_SecureChannel.cpp.
References DER_LEN_LONG_1, DER_LEN_LONG_2, and DER_LEN_LONG_FLAG.
Referenced by derSkipField(), and derWalkMfCert().

6.45.3.2 derSkipField()

```
bool derSkipField (
            const uint8_t * buf,
            uint16_t bufLen,
            uint16_t & pos)  [static]
```

Definition at line 767 of file CW_SecureChannel.cpp.
References derReadLength().
Referenced by derWalkMfCert().

6.45.3.3 derWalkMfCert()

```
bool derWalkMfCert (
            const uint8_t * buf,
            uint16_t bufLen,
            uint16_t & tbsMsgStart,
            uint16_t & tbsMsgLen,
            const uint8_t *& pubKey65Ptr,
            const uint8_t *& sigPtr,
            uint8_t & sigLen)  [static]
```

Definition at line 791 of file CW_SecureChannel.cpp.
References DER_BIT_UNUSED_ZERO, DER_EC_POINT_BYTES, DER_EC_UNCOMPRESSED, DER_TAG_BIT_STRING, DER_TAG_CTX0, DER_TAG_SEQUENCE, derReadLength(), and derSkipField().
Referenced by LLVMFuzzerTestOneInput(), and CW_SecureChannel::verifyCertificateChain().

6.45.4 Variable Documentation

6.45.4.1 s_apduBuf

```
uint8_t s_apduBuf[SEND_APDU_MAX_LEN]  [static]
```

Definition at line 60 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::aesCbcDecrypt(), and CW_SecureChannel::aesCbcEncrypt().

6.45.4.2 s_dataBuf

```
uint8_t s_dataBuf[ENC_BUF_MAX_LEN]  [static]
```
Definition at line 62 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::aesCbcDecrypt(), and CW_SecureChannel::aesCbcEncrypt().

6.45.4.3 s_macBuf

```
uint8_t s_macBuf[MAX_MAC_DATA_LEN]  [static]
```
Definition at line 61 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::aesCbcDecrypt(), and CW_SecureChannel::aesCbcEncrypt().

6.45.4.4 s_mfCertBuf

```
uint8_t s_mfCertBuf[CW_MANUF_CERT_MAX_BYTES]  [static]
```
Definition at line 65 of file CW_SecureChannel.cpp.
Referenced by CW_SecureChannel::preFetchManufacturerCert(), and CW_SecureChannel::verifyCertificateChain().

6.46 CW_SecureChannel.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
00018
00019 #include "CW_SecureChannel.h"
00020 #include "CW_Utils.h"
00021 #include "CW_TrustedKeys.h"
00022
00023 /*****************************************************************
00024  * Module-level constants
00025  *****************************************************************/
00026
00027 #define RESPONSE_GETCARDCERTIFICATE_IN_BYTES        148U
00028 /* SELECT AID: 1 (type) + 3 (ver) + 32 (status) + 2 (SW) = 38 bytes */
00029 #define RESPONSE_SELECT_IN_BYTES                     40U
00030 /* GET_MANUFACTURER_CERT: full DataOut up to certLen(2)+cert(411)+SW(2)=415 bytes; 420 for margin. */
00031 #define RESPONSE_GETMANUFACTURERCERT_PAGE_IN_BYTES  420U
00032 #define RESPONSE_OPENSECURECHANNEL_IN_BYTES      34U
00033 #define REQUEST_MUTUALLYAUTHENTICATE_IN_BYTES    69U
00034 #define RESPONSE_MUTUALLYAUTHENTICATE_IN_BYTES   66U
00035 #define RESPONSE_STATUS_WORDS_IN_BYTES            2U
00036
00037 #define OPENSECURECHANNEL_SALT_IN_BYTES   (RESPONSE_OPENSECURECHANNEL_IN_BYTES -
      RESPONSE_STATUS_WORDS_IN_BYTES)
00038 #define GETCARDCERTIFICATE_IN_BYTES       (RESPONSE_GETCARDCERTIFICATE_IN_BYTES -
      RESPONSE_STATUS_WORDS_IN_BYTES)
00039
00040 #define RANDOM_BYTES               8U
00041 #define COMMON_PAIRING_DATA        CW_PAIRING_DATA
00042 #define CLIENT_PRIVATE_KEY_SIZE   32U
00043 #define CLIENT_PUBLIC_KEY_SIZE    64U
00044 #define CARDEPHEMERALPUBKEY_SIZE  64U
00045 #define AES_BLOCK_SIZE            16U
00046 #define APDU_HEADER_LEN            (4U)
00047 #define APDU_LC_LEN                (1U)
00048 #define MAC_APDU_LEN              (12U)
00049 #define INPUT_BUFFER_LIMIT         (CW_USER_DATA_PAGE_SIZE)
00050 #define ENC_BUF_MAX_LEN            (INPUT_BUFFER_LIMIT + AES_BLOCK_SIZE)
00051 #define MAX_MAC_DATA_LEN           (APDU_HEADER_LEN + MAC_APDU_LEN + ENC_BUF_MAX_LEN)
00052 #define SEND_APDU_MAX_LEN          (APDU_HEADER_LEN + APDU_LC_LEN + AES_BLOCK_SIZE + ENC_BUF_MAX_LEN)
00053
00054 /* Enforce APDU fits within a single PN532 APDU (255 bytes max) */
00055 static_assert(APDU_HEADER_LEN + APDU_LC_LEN + AES_BLOCK_SIZE + ENC_BUF_MAX_LEN <= 255U,
00056               "CW_USER_DATA_PAGE_SIZE too large for PN532 single APDU transport");
00057
00058 /* Shared static crypto scratch buffers -- reuse is safe because decrypt is
00059  * always called from inside encrypt AFTER encrypt's large buffers are done. */
00060 static uint8_t s_apduBuf[SEND_APDU_MAX_LEN];  /* 245 bytes */
00061 static uint8_t s_macBuf [MAX_MAC_DATA_LEN];   /* 240 bytes */
00062 static uint8_t s_dataBuf[ENC_BUF_MAX_LEN];   /* 224 bytes */
00063
00064 /* Manufacturer certificate assembly buffer (used only during verifyCertificateChain). */
```

```
00065 static uint8_t s_mfCertBuf[CW_MANUF_CERT_MAX_BYTES];
00066
00067 /* DER TLV tag bytes */
00068 #define DER_TAG_SEQUENCE    (0x30U)  /* SEQUENCE (universal, constructed) */
00069 #define DER_TAG_BIT_STRING  (0x03U)  /* BIT STRING */
00070 #define DER_TAG_CTX0        (0xA0U)  /* [0] EXPLICIT -- version in v3 TBSCertificate */
00071
00072 /* DER length-field encoding */
00073 #define DER_LEN_LONG_FLAG   (0x80U)  /* set = long-form length */
00074 #define DER_LEN_LONG_1      (0x81U)  /* long form, 1 following byte */
00075 #define DER_LEN_LONG_2      (0x82U)  /* long form, 2 following bytes */
00076
00077 /* EC-point encoding */
00078 #define DER_EC_UNCOMPRESSED (0x04U)  /* uncompressed point prefix */
00079 #define DER_EC_POINT_BYTES  (65U)    /* 0x04 || X[32] || Y[32] */
00080 #define DER_BIT_UNUSED_ZERO (0x00U)  /* BIT STRING unused-bits field must be 0 */
00081
00082 /**********************************************************************
00083  * Constructor
00084  **********************************************************************/
00085
00086 // cppcheck-suppress misra-c2012-12.3 -- C++: member initializer-list commas are not the comma
      operator
00087 CW_SecureChannel::CW_SecureChannel(CW_NfcTransport& driver,
00088                                    CW_Logger& logger,
00089                                    CW_CryptoProvider& crypto,
00090                                    CW_Platform& platform)
00091     : _driver(driver), _logger(logger), _crypto(crypto), _platform(platform),
00092       _cachedMfCertLen(0U) {
00093     memset(_lastNonce, 0, sizeof(_lastNonce));
00094 }
00095
00096 /**********************************************************************
00097  * Transport delegation methods
00098  **********************************************************************/
00099
00100 bool CW_SecureChannel::begin() {
00101     return _driver.begin();
00102 }
00103
00104 bool CW_SecureChannel::inListPassiveTarget() {
00105     return _driver.inListPassiveTarget();
00106 }
00107
00108 void CW_SecureChannel::resetReader() {
00109     _driver.resetReader();
00110 }
00111
00112 bool CW_SecureChannel::printFirmwareVersion() {
00113     return _driver.printFirmwareVersion();
00114 }
00115
00116 /**********************************************************************
00117  * Private helpers
00118  **********************************************************************/
00119
00120 bool CW_SecureChannel::checkStatusWord(const uint8_t* response, uint16_t responseLength,
00121                                        uint8_t sw1Expected, uint8_t sw2Expected) {
00122     bool ret = false;
00123
00124     if ((response == NULL) || (responseLength < 2U)) {
00125 #if CW_DEBUG_LOGGING
00126         _logger.println(F("checkStatusWord: response too short."));
00127 #endif
00128     }
00129     else {
00130         uint8_t sw1 = response[responseLength - 2U];
00131         uint8_t sw2 = response[responseLength - 1U];
00132
00133         if ((sw1 == sw1Expected) && (sw2 == sw2Expected)) {
00134             ret = true;
00135         }
00136         else {
00137 #if CW_DEBUG_LOGGING
00138             _logger.print(F("SW: 0x"));
00139             if (sw1 < 16U) { _logger.print(F("0")); }
00140             _logger.print(sw1, HEX);
00141             _logger.print(F(" 0x"));
00142             if (sw2 < 16U) { _logger.print(F("0")); }
00143             _logger.println(sw2, HEX);
00144 #endif
00145         }
00146     }
00147
00148     return ret;
00149 }
00150
```

```
00151 /******************************************************************
00152  * Public methods
00153  ******************************************************************/
00154
00155 bool CW_SecureChannel::selectApdu() {
00156     bool ret = false;
00157
00158     uint8_t selectApduCmd[] = {
00159         0x00, 0xA4, 0x04, 0x00,
00160         0x07,
00161         0xA0, 0x00, 0x00, 0x10, 0x00, 0x01, 0x12
00162     };
00163
00164     uint8_t response[RESPONSE_SELECT_IN_BYTES];
00165     uint8_t responseLength = static_cast<uint8_t>(sizeof(response));
00166
00167     if (_driver.sendAPDU(selectApduCmd, sizeof(selectApduCmd), response, responseLength)) {
00168         if (checkStatusWord(response, responseLength, 0x90U, 0x00U)) {
00169             ret = true;
00170         } else {
00171 #if CW_DEBUG_LOGGING
00172             _logger.println(F("Select APDU failed."));
00173 #endif
00174         }
00175     } else {
00176 #if CW_DEBUG_LOGGING
00177         _logger.println(F("APDU select failed."));
00178 #endif
00179     }
00180
00181     return ret;
00182 }
00183
00184 bool CW_SecureChannel::getCardCertificate(uint8_t* cardCertificate, uint8_t& cardCertificateLength) {
00185     bool ret = false;
00186     uint8_t getCardCertificateResponse[RESPONSE_GETCARDCERTIFICATE_IN_BYTES];
00187     uint8_t getCardCertificateResponseLength =
    static_cast<uint8_t>(sizeof(getCardCertificateResponse));
00188
00189     if (cardCertificate != NULL) {
00190         uint8_t randomBytes[RANDOM_BYTES] = { 0U };
00191         if (!_crypto.random(randomBytes, RANDOM_BYTES)) {
00192 #if CW_DEBUG_LOGGING
00193             _logger.println(F("getCardCertificate: RNG failed."));
00194 #endif
00195             return false;
00196         }
00197
00198         /* Store nonce for replay check in verifyCertificateChain(). */
00199         (void)CW_Utils::safe_memcpy(_lastNonce, RANDOM_BYTES, randomBytes, RANDOM_BYTES);
00200
00201         uint8_t getCardCertificateApdu[] = {
00202             0x80, 0xF8, 0x00, 0x00, 0x08
00203         };
00204
00205         uint8_t fullApdu[sizeof(getCardCertificateApdu) + RANDOM_BYTES];
00206         (void)CW_Utils::safe_memcpy(fullApdu, sizeof(fullApdu), getCardCertificateApdu,
    sizeof(getCardCertificateApdu));
00207         (void)CW_Utils::safe_memcpy(fullApdu + sizeof(getCardCertificateApdu), RANDOM_BYTES,
    randomBytes, RANDOM_BYTES);
00208
00209         if (_driver.sendAPDU(fullApdu, sizeof(fullApdu),
00210                              getCardCertificateResponse, getCardCertificateResponseLength)) {
00211             if (checkStatusWord(getCardCertificateResponse, getCardCertificateResponseLength,
00212                                 0x90U, 0x00U)) {
00213                 cardCertificateLength = static_cast<uint8_t>(
00214                     static_cast<uint8_t>(getCardCertificateResponseLength -
    RESPONSE_STATUS_WORDS_IN_BYTES));
00215                 (void)CW_Utils::safe_memcpy(cardCertificate, GETCARDCERTIFICATE_IN_BYTES,
    getCardCertificateResponse, cardCertificateLength);
00216                 ret = true;
00217             } else {
00218 #if CW_DEBUG_LOGGING
00219                 _logger.println(F("getCardCertificate: bad SW."));
00220 #endif
00221             }
00222         } else {
00223 #if CW_DEBUG_LOGGING
00224             _logger.println(F("getCardCertificate APDU failed."));
00225 #endif
00226         }
00227     }
00228
00229     return ret;
00230 }
00231
00232 bool CW_SecureChannel::extractCardEphemeralKey(const uint8_t* cardCertificate,
```

```
00233                                                 uint8_t* cardEphemeralPubKey,
00234                                                 uint8_t* fullEphemeralPubKey65) {
00235     bool ret = false;
00236 
00237     if ((cardCertificate == NULL) || (cardEphemeralPubKey == NULL)) {
00238         ret = false;
00239     }
00240     else {
00241         const uint8_t keyStart = 1U + 8U; /* skip 'C' and nonce */
00242         const uint8_t fullKeyLength = 65U;
00243 
00244         /* Reject any key that is not an uncompressed point (0x04 prefix). */
00245         if (cardCertificate[keyStart] != 0x04U) {
00246             ret = false;
00247         }
00248         else {
00249             for (uint8_t i = 0U; i < fullKeyLength; i++) {
00250                 uint8_t b = cardCertificate[keyStart + i];
00251                 if (fullEphemeralPubKey65 != NULL) {
00252                     fullEphemeralPubKey65[i] = b;
00253                 }
00254                 if (i > 0U) {
00255                     cardEphemeralPubKey[i - 1U] = b;
00256                 }
00257             }
00258             ret = true;
00259         }
00260     }
00261 
00262     return ret;
00263 }
00264 
00265 bool CW_SecureChannel::openSecureChannel(uint8_t* salt,
00266                                          uint8_t* sessionPublicKey,
00267                                          uint8_t* sessionPrivateKey,
00268                                          CW_Curve sessionCurve) {
00269     bool ret = false;
00270 
00271     if (!_crypto.makeKey(sessionPublicKey, sessionPrivateKey, sessionCurve)) {
00272 #if CW_DEBUG_LOGGING
00273         _logger.println(F("ECC key generation failed."));
00274 #endif
00275     }
00276     else {
00277         uint8_t opcApduHeader[] = {
00278             0x80, 0x10, 0x00, 0x00, 0x41, 0x04
00279         };
00280 
00281         uint8_t fullApdu[sizeof(opcApduHeader) + CLIENT_PUBLIC_KEY_SIZE];
00282         (void)CW_Utils::safe_memcpy(fullApdu, sizeof(fullApdu), opcApduHeader, sizeof(opcApduHeader));
00283         (void)CW_Utils::safe_memcpy(fullApdu + sizeof(opcApduHeader), CLIENT_PUBLIC_KEY_SIZE,
    sessionPublicKey, CLIENT_PUBLIC_KEY_SIZE);
00284 
00285         uint8_t response[RESPONSE_OPENSECURECHANNEL_IN_BYTES];
00286         uint8_t responseLength = static_cast<uint8_t>(sizeof(response));
00287 
00288         if (_driver.sendAPDU(fullApdu, sizeof(fullApdu), response, responseLength)) {
00289             if (checkStatusWord(response, responseLength, 0x90U, 0x00U)) {
00290                 if (responseLength == static_cast<uint8_t>(RESPONSE_OPENSECURECHANNEL_IN_BYTES)) {
00291                     (void)CW_Utils::safe_memcpy(salt, OPENSECURECHANNEL_SALT_IN_BYTES, response,
    OPENSECURECHANNEL_SALT_IN_BYTES);
00292                     ret = true;
00293                 } else {
00294 #if CW_DEBUG_LOGGING
00295                     _logger.println(F("OpenSecureChannel: unexpected response size."));
00296 #endif
00297                 }
00298             } else {
00299 #if CW_DEBUG_LOGGING
00300                 _logger.println(F("OpenSecureChannel: bad SW."));
00301 #endif
00302             }
00303         } else {
00304 #if CW_DEBUG_LOGGING
00305             _logger.println(F("OpenSecureChannel APDU failed."));
00306 #endif
00307         }
00308     }
00309 
00310     return ret;
00311 }
00312 
00334 bool CW_SecureChannel::mutuallyAuthenticate(CW_SecureSession& session,
00335                                             const uint8_t* salt,
00336                                             uint8_t* clientPublicKey,
00337                                             const uint8_t* clientPrivateKey,
00338                                             CW_Curve sessionCurve,
```

```
00339                                           const uint8_t* cardEphemeralPubKey) {
00340     bool ret = false;
00341     uint8_t sharedSecret[32U] = { 0U };
00342     (void)clientPublicKey;
00343 
00344     if (!_crypto.ecdh(cardEphemeralPubKey, clientPrivateKey, sharedSecret, sessionCurve)) {
00345 #if CW_DEBUG_LOGGING
00346         _logger.println(F("ECDH failed."));
00347 #endif
00348     }
00349     else {
00350         uint8_t concat[32U + sizeof(COMMON_PAIRING_DATA) - 1U + 32U] = { 0U };
00351         uint8_t sha512Output[64U] = { 0U };
00352         const size_t pairingKeyLen = sizeof(COMMON_PAIRING_DATA) - 1U;
00353         const size_t concatLen     = 32U + pairingKeyLen + 32U;
00354 
00355         (void)CW_Utils::safe_memcpy(concat, sizeof(concat), sharedSecret, 32U);
00356         (void)CW_Utils::safe_memcpy(concat + 32U, sizeof(concat) - 32U, reinterpret_cast<const
      uint8_t*>(COMMON_PAIRING_DATA), pairingKeyLen);
00357         (void)CW_Utils::safe_memcpy(concat + 32U + pairingKeyLen, 32U, salt, 32U);
00358 
00359         bool sha512Ok = _crypto.sha512(concat, concatLen, sha512Output);
00360 
00361         if (!sha512Ok) {
00362             CW_Utils::secure_wipe(sharedSecret, sizeof(sharedSecret));
00363             CW_Utils::secure_wipe(sha512Output, sizeof(sha512Output));
00364             CW_Utils::secure_wipe(concat, sizeof(concat));
00365             return false;
00366         }
00367 
00368         (void)CW_Utils::safe_memcpy(session.aesKey, CW_AESKEY_SIZE, sha512Output, CW_AESKEY_SIZE);
00369         (void)CW_Utils::safe_memcpy(session.macKey, CW_MACKEY_SIZE, sha512Output + CW_AESKEY_SIZE,
      CW_MACKEY_SIZE);
00370 
00371         uint8_t iv_opc[AES_BLOCK_SIZE] = { 0U };
00372         uint8_t mac_iv[AES_BLOCK_SIZE] = { 0U };
00373         memset(iv_opc, 0x01U, AES_BLOCK_SIZE);
00374 
00375         uint8_t RNG_data[32U] = { 0U };
00376         if (!_crypto.random(RNG_data, sizeof(RNG_data))) {
00377 #if CW_DEBUG_LOGGING
00378             _logger.println(F("RNG failed."));
00379 #endif
00380             session.clear();
00381             CW_Utils::secure_wipe(sharedSecret, sizeof(sharedSecret));
00382             CW_Utils::secure_wipe(sha512Output, sizeof(sha512Output));
00383             CW_Utils::secure_wipe(concat, sizeof(concat));
00384             return false;
00385         }
00386 
00387         /* Encrypt random data with Kenc (Bit padding) */
00388         uint8_t ciphertextOPC[48U] = { 0U };
00389         uint16_t cipherLength = _crypto.aesCbcEncrypt(RNG_data, sizeof(RNG_data),
00390                                                       ciphertextOPC,
00391                                                       session.aesKey, sizeof(session.aesKey),
00392                                                       iv_opc, true);
00393 
00394         /* Compute MAC over APDU header + ciphertext (Null padding) */
00395         uint8_t opcApduHeader[APDU_HEADER_LEN + APDU_LC_LEN] = {
00396             0x80U, 0x11U, 0x00U, 0x00U,
00397             (uint8_t)(cipherLength + AES_BLOCK_SIZE)
00398         };
00399         uint8_t MAC_apduHeader[AES_BLOCK_SIZE] = { 0U };
00400         (void)CW_Utils::safe_memcpy(MAC_apduHeader, sizeof(MAC_apduHeader), opcApduHeader,
      sizeof(opcApduHeader));
00401 
00402         size_t  MAC_data_length = sizeof(MAC_apduHeader) + cipherLength;
00403         uint8_t MAC_data[64U] = { 0U };
00404         uint8_t ciphertextMACLong[64U] = { 0U };
00405 
00406         if (MAC_data_length > sizeof(MAC_data)) {
00407             session.clear();
00408             CW_Utils::secure_wipe(sharedSecret, sizeof(sharedSecret));
00409             CW_Utils::secure_wipe(sha512Output, sizeof(sha512Output));
00410             CW_Utils::secure_wipe(concat, sizeof(concat));
00411             CW_Utils::secure_wipe(RNG_data, sizeof(RNG_data));
00412             return false;
00413         }
00414 
00415         (void)CW_Utils::safe_memcpy(MAC_data, sizeof(MAC_data), MAC_apduHeader,
      sizeof(MAC_apduHeader));
00416         (void)CW_Utils::safe_memcpy(MAC_data + sizeof(MAC_apduHeader), sizeof(MAC_data) -
      sizeof(MAC_apduHeader), ciphertextOPC, cipherLength);
00417 
00418         uint16_t encryptedLengthMAC = _crypto.aesCbcEncrypt(MAC_data, (uint16_t)MAC_data_length,
00419                                                             ciphertextMACLong,
00420                                                             session.macKey, sizeof(session.macKey),
```

```
00421                                                       mac_iv, false);
00422
00423         uint8_t MAC_value[AES_BLOCK_SIZE] = { 0U };
00424         uint8_t macOffset = (uint8_t)(encryptedLengthMAC - AES_BLOCK_SIZE);
00425         (void)CW_Utils::safe_memcpy(MAC_value, sizeof(MAC_value), ciphertextMACLong + macOffset,
      AES_BLOCK_SIZE);
00426
00427         /* Forge MUTUALLY AUTHENTICATE APDU */
00428         uint8_t sendApduOpc[REQUEST_MUTUALLYAUTHENTICATE_IN_BYTES] = { 0U };
00429         uint16_t offset = 0U;
00430         (void)CW_Utils::safe_memcpy(sendApduOpc + offset, sizeof(sendApduOpc) -
      static_cast<size_t>(offset), opcApduHeader, sizeof(opcApduHeader));
00431         offset += sizeof(opcApduHeader);
00432         (void)CW_Utils::safe_memcpy(sendApduOpc + offset, sizeof(sendApduOpc) -
      static_cast<size_t>(offset), MAC_value, sizeof(MAC_value));
00433         offset += sizeof(MAC_value);
00434         (void)CW_Utils::safe_memcpy(sendApduOpc + offset, sizeof(sendApduOpc) -
      static_cast<size_t>(offset), ciphertextOPC, cipherLength);
00435
00436         uint8_t response[255U] = { 0U };
00437         uint8_t responseLength = static_cast<uint8_t>(sizeof(response));
00438
00439         if (_driver.sendAPDU(sendApduOpc, sizeof(sendApduOpc), response, responseLength)) {
00440             if (checkStatusWord(response, responseLength, 0x90U, 0x00U)) {
00441                 if (responseLength == static_cast<uint8_t>(RESPONSE_MUTUALLYAUTHENTICATE_IN_BYTES)) {
00442                     (void)CW_Utils::safe_memcpy(session.iv, CW_IV_SIZE, response, CW_IV_SIZE);
00443                     ret = true;
00444                 } else {
00445 #if CW_DEBUG_LOGGING
00446                     _logger.println(F("MutualAuth: unexpected response size."));
00447 #endif
00448                 }
00449             } else {
00450 #if CW_DEBUG_LOGGING
00451                 _logger.println(F("MutualAuth: bad SW."));
00452 #endif
00453             }
00454         } else {
00455 #if CW_DEBUG_LOGGING
00456             _logger.println(F("MutualAuth APDU failed."));
00457 #endif
00458         }
00459
00460         /* Secure cleanup */
00461         CW_Utils::secure_wipe(sharedSecret, sizeof(sharedSecret));
00462         CW_Utils::secure_wipe(sha512Output, sizeof(sha512Output));
00463         CW_Utils::secure_wipe(concat, sizeof(concat));
00464         CW_Utils::secure_wipe(RNG_data, sizeof(RNG_data));
00465         CW_Utils::secure_wipe(ciphertextOPC, sizeof(ciphertextOPC));
00466         CW_Utils::secure_wipe(MAC_data, sizeof(MAC_data));
00467
00468         /* If the APDU exchange failed after session keys were written, clear
00469          * them now to prevent a half-initialised session from being used (CRIT-04). */
00470         if (!ret) {
00471             session.clear();
00472         }
00473     }
00474
00475     return ret;
00476 }
00477
00501 bool CW_SecureChannel::aesCbcEncrypt(CW_SecureSession& session,
00502                                      const uint8_t apdu[], uint16_t apduLength,
00503                                      const uint8_t data[], uint16_t dataLength,
00504                                      uint8_t* decryptedOutput, uint16_t* decryptedOutputLength) {
00505     bool ret = false;
00506
00507     /* Reject payloads that would overflow s_dataBuf (MED-01). */
00508     if (dataLength > INPUT_BUFFER_LIMIT) {
00509 #if CW_DEBUG_LOGGING
00510         _logger.println(F("Error: data too large for encryption buffer."));
00511 #endif
00512         return false;
00513     }
00514
00515     /* 1. Encrypt data with Kenc (Bit padding) */
00516     uint16_t encryptedLength = _crypto.aesCbcEncrypt(data, dataLength, s_dataBuf,
00517                                                      session.aesKey, sizeof(session.aesKey),
00518                                                      session.iv, true);
00519
00520     uint16_t lcValue = encryptedLength + (uint16_t)AES_BLOCK_SIZE;
00521     /* lcValue must fit in uint8_t: with INPUT_BUFFER_LIMIT=208, max encryptedLength=224,
00522      * so max lcValue=240. The static_assert above also caps APDU at 255 bytes (MED-03). */
00523     if (lcValue > 0xFFU) {
00524 #if CW_DEBUG_LOGGING
00525         _logger.println(F("Error: lcValue overflow -- payload too large."));
00526 #endif
```

```
00527         CW_Utils::secure_wipe(s_dataBuf, sizeof(s_dataBuf));
00528         return false;
00529     }
00530     uint8_t macApdu[MAC_APDU_LEN] = { 0U };
00531     /* MED-03: Single-byte length encoding and no direction byte are intentional --
00532      * this CBC-MAC construction matches the Cryptnox SCCP card firmware spec.
00533      * Changing to AES-CMAC, wider encoding, or adding a direction byte would
00534      * break the card protocol. lcValue overflow is prevented by the
00535      * INPUT_BUFFER_LIMIT precondition above (dataLength <= 208 → lcValue <= 240). */
00536     macApdu[0U] = (uint8_t)lcValue;
00537
00538     uint16_t macDataLength = apduLength + sizeof(macApdu) + encryptedLength;
00539     if (macDataLength > MAX_MAC_DATA_LEN) {
00540 #if CW_DEBUG_LOGGING
00541         _logger.println(F("Error: MAC data length exceeds buffer."));
00542 #endif
00543         CW_Utils::secure_wipe(s_dataBuf, sizeof(s_dataBuf));
00544         return false;
00545     }
00546
00547     /* 2. Build MAC input: APDU header || LC block || ciphertext */
00548     uint16_t offset = 0U;
00549     (void)CW_Utils::safe_memcpy(s_macBuf, sizeof(s_macBuf), apdu, apduLength);
00550     offset += apduLength;
00551     (void)CW_Utils::safe_memcpy(s_macBuf + offset, sizeof(s_macBuf) - static_cast<size_t>(offset),
    macApdu, sizeof(macApdu));
00552     offset += sizeof(macApdu);
00553     (void)CW_Utils::safe_memcpy(s_macBuf + offset, sizeof(s_macBuf) - static_cast<size_t>(offset),
    s_dataBuf, encryptedLength);
00554
00555     uint8_t macIv[AES_BLOCK_SIZE] = { 0U };
00556     uint16_t macEncryptedLength = _crypto.aesCbcEncrypt(s_macBuf, macDataLength, s_apduBuf,
00557                                                         session.macKey, sizeof(session.macKey),
00558                                                         macIv, false);
00559
00560     uint8_t macValue[AES_BLOCK_SIZE] = { 0U };
00561     uint16_t macOffset = macEncryptedLength - AES_BLOCK_SIZE;
00562     (void)CW_Utils::safe_memcpy(macValue, sizeof(macValue), s_apduBuf + macOffset, AES_BLOCK_SIZE);
00563
00564     /* 3. Build send APDU: header || Lc || MAC || ciphertext */
00565     const uint8_t lc = (uint8_t)lcValue;
00566     uint8_t sendApduLength = (uint8_t)(apduLength + APDU_LC_LEN + sizeof(macValue) + encryptedLength);
00567     if (sendApduLength > SEND_APDU_MAX_LEN) {
00568 #if CW_DEBUG_LOGGING
00569         _logger.println(F("Error: Send APDU length exceeds buffer."));
00570 #endif
00571         CW_Utils::secure_wipe(s_dataBuf, sizeof(s_dataBuf));
00572         CW_Utils::secure_wipe(s_macBuf,  sizeof(s_macBuf));
00573         return false;
00574     }
00575
00576     offset = 0U;
00577     (void)CW_Utils::safe_memcpy(s_apduBuf, sizeof(s_apduBuf), apdu, apduLength);
00578     offset += apduLength;
00579     s_apduBuf[offset] = lc;
00580     offset += APDU_LC_LEN;
00581     (void)CW_Utils::safe_memcpy(s_apduBuf + offset, sizeof(s_apduBuf) - static_cast<size_t>(offset),
    macValue, sizeof(macValue));
00582     offset += sizeof(macValue);
00583     (void)CW_Utils::safe_memcpy(s_apduBuf + offset, sizeof(s_apduBuf) - static_cast<size_t>(offset),
    s_dataBuf, encryptedLength);
00584
00585     /* 4. Send APDU */
00586     uint8_t response[255U] = { 0U };
00587     uint8_t responseLength = static_cast<uint8_t>(sizeof(response));
00588
00589     if (_driver.sendAPDU(s_apduBuf, sendApduLength, response, responseLength)) {
00590         if (checkStatusWord(response, responseLength, 0x90U, 0x00U)) {
00591             /* Update session.iv ONLY after the response MAC is verified (HIGH-01).
00592              * Moving it before aesCbcDecrypt would let an attacker-chosen IV
00593              * desynchronise the rolling-IV state even on MAC failure. */
00594             ret = aesCbcDecrypt(session, response, static_cast<size_t>(responseLength), macValue,
00595                                 decryptedOutput, decryptedOutputLength);
00596             if (ret) {
00597                 (void)CW_Utils::safe_memcpy(session.iv, CW_IV_SIZE, response, CW_IV_SIZE);
00598             } else {
00599                 session.clear();
00600             }
00601         } else if (responseLength >= 2U) {
00602             /* Card-level error: surface the SW so the caller can diagnose
00603              * common cases (e.g. 0x63Cn = wrong PIN with n retries left). */
00604 #if CW_DEBUG_LOGGING
00605             _logger.print(F("Secured APDU: bad SW 0x"));
00606             if (response[responseLength - 2U] < 0x10U) { _logger.print(F("0")); }
00607             _logger.print(response[responseLength - 2U], HEX);
00608             if (response[responseLength - 1U] < 0x10U) { _logger.print(F("0")); }
00609             _logger.println(response[responseLength - 1U], HEX);
```

```
00610 #endif
00611         }
00612     } else {
00613 #if CW_DEBUG_LOGGING
00614         _logger.println(F("Secured APDU failed."));
00615 #endif
00616     }
00617 
00618     /* Wipe plaintext scratch buffers so they do not persist in .bss (MED-04). */
00619     CW_Utils::secure_wipe(s_dataBuf, sizeof(s_dataBuf));
00620     CW_Utils::secure_wipe(s_macBuf,  sizeof(s_macBuf));
00621 
00622     return ret;
00623 }
00624 
00625 bool CW_SecureChannel::aesCbcDecrypt(const CW_SecureSession& session,
00626                                      uint8_t* response, size_t response_len,
00627                                      uint8_t* mac_value,
00628                                      uint8_t* decryptedOutput, uint16_t* decryptedOutputLength) {
00629     /* Precondition: response must hold at least MAC(16) + 1 ciphertext byte + SW(2) (HIGH-02).
00630      * Without this check, response_len < 18 causes size_t underflow in the subtractions below. */
00631     if ((response == NULL) || (response_len < (size_t)(AES_BLOCK_SIZE + 2U + 1U))) {
00632         return false;
00633     }
00634 
00635     /* Response layout: MAC(16) || cipherText(N) || SW1(1) || SW2(1) */
00636     uint8_t rep_mac[AES_BLOCK_SIZE];
00637     (void)CW_Utils::safe_memcpy(rep_mac, sizeof(rep_mac), response, AES_BLOCK_SIZE);
00638     uint8_t* rep_data  = response + AES_BLOCK_SIZE;
00639     size_t totalDataLen = response_len - 2U;
00640     size_t cipherLen    = totalDataLen - AES_BLOCK_SIZE;
00641 
00642     if (mac_value == NULL) {
00643         return false;
00644     }
00645 
00646     /* Verify MAC: AES-CBC-MAC over [length_header(16)] || [all_ciphertext] */
00647     size_t macInputLen = AES_BLOCK_SIZE + cipherLen;
00648     if (macInputLen > sizeof(s_macBuf)) {
00649 #if CW_DEBUG_LOGGING
00650         _logger.println(F("Error: Response too large for MAC verification."));
00651 #endif
00652         return false;
00653     }
00654 
00655     memset(s_macBuf, 0U, AES_BLOCK_SIZE);
00656     s_macBuf[0] = (uint8_t)totalDataLen;
00657     (void)CW_Utils::safe_memcpy(s_macBuf + AES_BLOCK_SIZE, sizeof(s_macBuf) - AES_BLOCK_SIZE,
      rep_data, cipherLen);
00658 
00659     uint8_t mac_iv[AES_BLOCK_SIZE] = { 0U };
00660     uint16_t macEncryptedLength = _crypto.aesCbcEncrypt(s_macBuf, (uint16_t)macInputLen, s_apduBuf,
00661                                                         session.macKey, sizeof(session.macKey),
00662                                                         mac_iv, false);
00663 
00664     uint8_t recomputedMacValue[AES_BLOCK_SIZE] = { 0U };
00665     uint16_t macOffset = macEncryptedLength - AES_BLOCK_SIZE;
00666     (void)CW_Utils::safe_memcpy(recomputedMacValue, sizeof(recomputedMacValue), s_apduBuf + macOffset,
      AES_BLOCK_SIZE);
00667 
00668     if (!CW_Utils::secure_compare(rep_mac, recomputedMacValue, AES_BLOCK_SIZE)) {
00669 #if CW_DEBUG_LOGGING
00670         _logger.println(F("MAC mismatch."));
00671 #endif
00672         CW_Utils::secure_wipe(s_macBuf, sizeof(s_macBuf));
00673         return false;
00674     }
00675 
00676     /* Decrypt ciphertext using mac_value as IV (Bit padding removal) */
00677     uint16_t decryptedDataLength = _crypto.aesCbcDecrypt(rep_data, (uint16_t)cipherLen, s_dataBuf,
00678                                                          session.aesKey, sizeof(session.aesKey),
00679                                                          mac_value, true);
00680 
00681     bool ret = false;
00682 
00683     if (decryptedDataLength < 2U) {
00684 #if CW_DEBUG_LOGGING
00685         _logger.println(F("Error: Decoded data too short."));
00686 #endif
00687     }
00688     else if (decryptedDataLength > sizeof(s_dataBuf)) {
00689 #if CW_DEBUG_LOGGING
00690         _logger.println(F("Error: Decoded data length exceeds buffer."));
00691 #endif
00692     }
00693     else {
00694         uint8_t innerSW1 = s_dataBuf[decryptedDataLength - 2U];
```

```
00695         uint8_t innerSW2 = s_dataBuf[decryptedDataLength - 1U];
00696         uint16_t payloadLength = decryptedDataLength - 2U;
00697 
00698         if ((innerSW1 != 0x90U) || (innerSW2 != 0x00U)) {
00699 #if CW_DEBUG_LOGGING
00700             _logger.print(F("Card error SW: 0x"));
00701             if (innerSW1 < 0x10U) { _logger.print(F("0")); }
00702             _logger.print(innerSW1, HEX);
00703             _logger.print(F(" 0x"));
00704             if (innerSW2 < 0x10U) { _logger.print(F("0")); }
00705             _logger.println(innerSW2, HEX);
00706 #endif
00707         }
00708         else {
00709             ret = true;
00710         }
00711 
00712         if ((decryptedOutput != NULL) && (decryptedOutputLength != NULL)) {
00713             (void)CW_Utils::safe_memcpy(decryptedOutput, sizeof(s_dataBuf), s_dataBuf, payloadLength);
00714             *decryptedOutputLength = payloadLength;
00715         }
00716     }
00717 
00718     /* Wipe plaintext scratch buffers (MED-04). */
00719     CW_Utils::secure_wipe(s_dataBuf, sizeof(s_dataBuf));
00720     CW_Utils::secure_wipe(s_macBuf,  sizeof(s_macBuf));
00721 
00722     return ret;
00723 }
00724 
00725 /**************************************************************************
00726  * Certificate verification -- static helpers
00727  **************************************************************************/
00728 
00729 
00730 /* Read the DER length at buf[*pos]; advance *pos past the length bytes.
00731  * Supports short form and long form with 1 or 2 extra bytes only. */
00732 static bool derReadLength(const uint8_t* buf, uint16_t bufLen,
00733                           uint16_t& pos, uint16_t& fieldLen) {
00734     bool    ok    = false;
00735     fieldLen      = 0U;
00736 
00737     if ((buf != NULL) && (pos < bufLen)) {
00738         uint8_t b = buf[pos];
00739         pos += 1U;
00740 
00741         if ((b & DER_LEN_LONG_FLAG) == 0U) {
00742             fieldLen = static_cast<uint16_t>(b);
00743             ok = true;
00744         } else if (b == DER_LEN_LONG_1) {
00745             if (pos < bufLen) {
00746                 fieldLen = static_cast<uint16_t>(buf[pos]);
00747                 pos += 1U;
00748                 ok = true;
00749             }
00750         } else if (b == DER_LEN_LONG_2) {
00751             if ((pos + 1U) < bufLen) {
00752                 fieldLen  = static_cast<uint16_t>(
00753                                 static_cast<uint16_t>(buf[pos]) << 8U);
00754                 fieldLen |= static_cast<uint16_t>(buf[pos + 1U]);
00755                 pos += 2U;
00756                 ok = true;
00757             }
00758         } else {
00759             ok = false; /* indefinite form or > 2 extra bytes -- unsupported */
00760         }
00761     }
00762 
00763     return ok;
00764 }
00765 
00766 /* Skip one complete DER TLV (tag byte + length bytes + value) at buf[*pos]. */
00767 static bool derSkipField(const uint8_t* buf, uint16_t bufLen, uint16_t& pos) {
00768     bool ok = false;
00769 
00770     if ((buf != NULL) && (pos < bufLen)) {
00771         uint16_t contentLen = 0U;
00772         pos += 1U; /* skip tag byte */
00773         if (derReadLength(buf, bufLen, pos, contentLen)) {
00774             if ((pos + contentLen) <= bufLen) {
00775                 pos += contentLen;
00776                 ok   = true;
00777             }
00778         }
00779     }
00780 
00781     return ok;
```

```
00782 }
00783
00784 /* Walk a DER X.509 Certificate to extract -- without any byte-pattern search:
00785  *   tbsMsgStart  offset of TBSCertificate SEQUENCE tag inside buf
00786  *   tbsMsgLen    total byte count of TBSCertificate (tag + length + content)
00787  *   pubKey65Ptr  pointer to 65-byte uncompressed EC point (0x04 || X || Y)
00788  *   sigPtr       pointer to the DER ECDSA signature bytes
00789  *   sigLen       byte count of the DER ECDSA signature
00790  * Returns false on any format or bounds error. */
00791 static bool derWalkMfCert(const uint8_t* buf, uint16_t bufLen,
00792                           uint16_t& tbsMsgStart, uint16_t& tbsMsgLen,
00793                           const uint8_t*& pubKey65Ptr,
00794                           const uint8_t*& sigPtr, uint8_t& sigLen) {
00795     bool     ok             = true;
00796     uint16_t pos            = 0U;
00797     uint16_t certContentLen = 0U;
00798     uint16_t tbsContentLen  = 0U;
00799     uint16_t tbsEnd         = 0U;
00800     uint16_t spkiContentLen = 0U;
00801     uint16_t bsLen          = 0U;
00802
00803     tbsMsgStart = 0U;
00804     tbsMsgLen   = 0U;
00805     pubKey65Ptr = NULL;
00806     sigPtr      = NULL;
00807     sigLen      = 0U;
00808
00809     if ((buf == NULL) || (bufLen == 0U)) {
00810         ok = false;
00811     }
00812
00813     /* - outer Certificate SEQUENCE - */
00814     if (ok) {
00815         if (buf[pos] != DER_TAG_SEQUENCE) {
00816             ok = false;
00817         }
00818     }
00819     if (ok) {
00820         pos += 1U;
00821         if (!derReadLength(buf, bufLen, pos, certContentLen)) {
00822             ok = false;
00823         }
00824     }
00825     if (ok) {
00826         if ((pos + certContentLen) > bufLen) {
00827             ok = false;
00828         }
00829     }
00830
00831     /* - TBSCertificate SEQUENCE (first child) - */
00832     if (ok) {
00833         if (buf[pos] != DER_TAG_SEQUENCE) {
00834             ok = false;
00835         }
00836     }
00837     if (ok) {
00838         tbsMsgStart = pos;
00839         pos += 1U;
00840         if (!derReadLength(buf, bufLen, pos, tbsContentLen)) {
00841             ok = false;
00842         }
00843     }
00844     if (ok) {
00845         tbsMsgLen = (pos - tbsMsgStart) + tbsContentLen;
00846         tbsEnd    = pos + tbsContentLen;
00847         if (tbsEnd > bufLen) {
00848             ok = false;
00849         }
00850     }
00851
00852     /* - Walk TBSCertificate fields in order - */
00853
00854     /* [0] EXPLICIT version -- present in X.509 v3 */
00855     if (ok && (pos < tbsEnd)) {
00856         if (buf[pos] == DER_TAG_CTX0) {
00857             if (!derSkipField(buf, tbsEnd, pos)) {
00858                 ok = false;
00859             }
00860         }
00861     }
00862
00863     /* serialNumber INTEGER */
00864     if (ok) {
00865         if (!derSkipField(buf, tbsEnd, pos)) {
00866             ok = false;
00867         }
00868     }
```

```
00869
00870     /* signature AlgorithmIdentifier SEQUENCE */
00871     if (ok) {
00872         if (!derSkipField(buf, tbsEnd, pos)) {
00873             ok = false;
00874         }
00875     }
00876
00877     /* issuer Name SEQUENCE */
00878     if (ok) {
00879         if (!derSkipField(buf, tbsEnd, pos)) {
00880             ok = false;
00881         }
00882     }
00883
00884     /* validity SEQUENCE */
00885     if (ok) {
00886         if (!derSkipField(buf, tbsEnd, pos)) {
00887             ok = false;
00888         }
00889     }
00890
00891     /* subject Name SEQUENCE */
00892     if (ok) {
00893         if (!derSkipField(buf, tbsEnd, pos)) {
00894             ok = false;
00895         }
00896     }
00897
00898     /* - SubjectPublicKeyInfo SEQUENCE - */
00899     if (ok) {
00900         if (pos >= tbsEnd) {
00901             ok = false;
00902         }
00903     }
00904     if (ok) {
00905         if (buf[pos] != DER_TAG_SEQUENCE) {
00906             ok = false;
00907         }
00908     }
00909     if (ok) {
00910         pos += 1U;
00911         if (!derReadLength(buf, bufLen, pos, spkiContentLen)) {
00912             ok = false;
00913         }
00914     }
00915     if (ok) {
00916         if ((pos + spkiContentLen) > bufLen) {
00917             ok = false;
00918         }
00919     }
00920
00921     /* Skip AlgorithmIdentifier SEQUENCE inside SubjectPublicKeyInfo */
00922     if (ok) {
00923         if (!derSkipField(buf, bufLen, pos)) {
00924             ok = false;
00925         }
00926     }
00927
00928     /* subjectPublicKey BIT STRING */
00929     if (ok) {
00930         if (pos >= bufLen) {
00931             ok = false;
00932         }
00933     }
00934     if (ok) {
00935         if (buf[pos] != DER_TAG_BIT_STRING) {
00936             ok = false;
00937         }
00938     }
00939     if (ok) {
00940         pos += 1U;
00941         if (!derReadLength(buf, bufLen, pos, bsLen)) {
00942             ok = false;
00943         }
00944     }
00945     if (ok) {
00946         if ((pos + bsLen) > bufLen) {
00947             ok = false;
00948         }
00949     }
00950     if (ok) {
00951         if (bsLen < (1U + static_cast<uint16_t>(DER_EC_POINT_BYTES))) {
00952             ok = false; /* too short: unused-bits byte + 65-byte EC point */
00953         }
00954     }
00955     if (ok) {
```

```
00956         if (buf[pos] != DER_BIT_UNUSED_ZERO) {
00957             ok = false; /* unused bits must be 0 */
00958         } else if (buf[pos + 1U] != DER_EC_UNCOMPRESSED) {
00959             ok = false; /* must be an uncompressed EC point */
00960         } else {
00961             pubKey65Ptr = buf + pos + 1U; /* points to: 0x04 || X[32] || Y[32] */
00962         }
00963     }
00964
00965     /* Jump to end of TBSCertificate, then skip signatureAlgorithm SEQUENCE */
00966     if (ok) {
00967         pos = tbsEnd;
00968         if (!derSkipField(buf, bufLen, pos)) {
00969             ok = false;
00970         }
00971     }
00972
00973     /* - signatureValue BIT STRING (third child of Certificate) - */
00974     if (ok) {
00975         if (pos >= bufLen) {
00976             ok = false;
00977         }
00978     }
00979     if (ok) {
00980         if (buf[pos] != DER_TAG_BIT_STRING) {
00981             ok = false;
00982         }
00983     }
00984     if (ok) {
00985         pos += 1U;
00986         if (!derReadLength(buf, bufLen, pos, bsLen)) {
00987             ok = false;
00988         }
00989     }
00990     if (ok) {
00991         if ((pos + bsLen) > bufLen) {
00992             ok = false;
00993         }
00994     }
00995     if (ok) {
00996         if (bsLen < 2U) {
00997             ok = false; /* need unused-bits byte + at least 1 signature byte */
00998         }
00999     }
01000     if (ok) {
01001         if (buf[pos] != DER_BIT_UNUSED_ZERO) {
01002             ok = false; /* unused bits must be 0 */
01003         }
01004     }
01005     if (ok) {
01006         uint16_t rawSigLen = bsLen - 1U;
01007         if (rawSigLen > 255U) {
01008             ok = false;
01009         } else {
01010             sigPtr = buf + pos + 1U; /* DER ECDSA signature, after unused-bits byte */
01011             sigLen = static_cast<uint8_t>(rawSigLen);
01012         }
01013     }
01014
01015     return ok;
01016 }
01017
01018 bool CW_SecureChannel::parseDerSigToRaw(const uint8_t* der, uint8_t derLen,
01019                                         uint8_t* raw64) {
01020     bool ret = false;
01021
01022     if ((der != NULL) && (raw64 != NULL) && (derLen >= 6U) && (der[0] == 0x30U)) {
01023         /* Validate outer SEQUENCE length against actual buffer (HIGH-04). */
01024         if ((uint8_t)(der[1] + 2U) > derLen) {
01025             return false;
01026         }
01027
01028         uint8_t pos = 2U;  /* skip SEQUENCE tag + length */
01029
01030         if (der[pos] == 0x02U) {
01031             pos++;
01032             uint8_t rLen = der[pos];
01033             pos++;
01034             /* Reject malformed r -- DER r is at most 33 bytes (32 + 1 zero pad) (HIGH-04). */
01035             if (rLen > 33U) {
01036                 return false;
01037             }
01038             if ((pos + rLen) <= derLen) {
01039                 const uint8_t* rPtr = der + pos;
01040                 pos += rLen;
01041
01042                 if ((pos < derLen) && (der[pos] == 0x02U)) {
```

```
01043                     pos++;
01044                     uint8_t sLen = der[pos];
01045                     pos++;
01046                     /* Reject malformed s (HIGH-04). */
01047                     if (sLen > 33U) {
01048                         return false;
01049                     }
01050                     if ((pos + sLen) <= derLen) {
01051                         /* M-06: verify sum of both INTEGER fields matches the outer SEQUENCE length.
01052                          * Trailing garbage after s would indicate malformed/crafted DER. */
01053                         if ((pos + sLen) != (uint8_t)(2U + der[1])) {
01054                             return false;
01055                         }
01056                         const uint8_t* sPtr = der + pos;
01057 
01058                         memset(raw64, 0U, 64U);
01059 
01060                         /* r: strip optional leading 0x00 padding byte */
01061                         if ((rLen == 33U) && (rPtr[0] == 0x00U)) { rPtr++; rLen = 32U; }
01062                         if (rLen <= 32U) {
01063                             (void)CW_Utils::safe_memcpy(raw64 + (32U - rLen), static_cast<size_t>(32U
      + rLen), rPtr, rLen);
01064                         }
01065 
01066                         /* s: strip optional leading 0x00 padding byte */
01067                         if ((sLen == 33U) && (sPtr[0] == 0x00U)) { sPtr++; sLen = 32U; }
01068                         if (sLen <= 32U) {
01069                             (void)CW_Utils::safe_memcpy(raw64 + 32U + (32U - sLen),
      static_cast<size_t>(sLen), sPtr, sLen);
01070                         }
01071 
01072                         ret = true;
01073                     }
01074                 }
01075             }
01076         }
01077     }
01078 
01079     return ret;
01080 }
01081 
01082 bool CW_SecureChannel::verifyEcdsaSha256(const uint8_t* pubKey64,
01083                                          const uint8_t* message, uint16_t msgLen,
01084                                          const uint8_t* derSig, uint8_t derSigLen) {
01085     bool result = false;
01086     uint8_t hash[32U] = { 0U };
01087     uint8_t rawSig[64U] = { 0U };
01088 
01089     bool hashOk = _crypto.sha256(message, msgLen, hash);
01090 
01091     if (hashOk && parseDerSigToRaw(derSig, derSigLen, rawSig)) {
01092         result = _crypto.ecdsaVerify(pubKey64, hash, sizeof(hash), rawSig, CW_CURVE_SECP256R1);
01093     }
01094 
01095     return result;
01096 }
01097 
01098 bool CW_SecureChannel::getManufacturerCertificate(uint8_t* cert, uint16_t& certLen) {
01099     bool ret = false;
01100     certLen = 0U;
01101 
01102     if (cert != NULL) {
01103         const uint8_t APDU_P2_IDX = 3U;  /* P2 field offset in ISO 7816-4 APDU header */
01104         uint8_t apdu[5U] = { 0x80U, 0xF7U, 0x00U, 0x00U, 0x00U };
01105         uint8_t response[RESPONSE_GETMANUFACTURERCERT_PAGE_IN_BYTES];
01106         uint16_t responseLen = static_cast<uint16_t>(sizeof(response));
01107 
01108         if (!_driver.sendAPDULarge(apdu, static_cast<uint8_t>(sizeof(apdu)), response,
01109                                    responseLen)) {
01110 #if CW_DEBUG_LOGGING
01111             _logger.println(F("getManufacturerCertificate APDU failed."));
01112 #endif
01113             return false;
01114         }
01115         if (responseLen > static_cast<uint16_t>(sizeof(response))) {
01116 #if CW_DEBUG_LOGGING
01117             _logger.println(F("getManufacturerCertificate: driver reported overflow."));
01118 #endif
01119             return false;
01120         }
01121 
01122         if (checkStatusWord(response, responseLen, 0x90U, 0x00U)) {
01123 
01124             uint16_t dataBytes = static_cast<uint16_t>(
01125                 responseLen - static_cast<uint16_t>(RESPONSE_STATUS_WORDS_IN_BYTES));
01126 
01127             if (dataBytes >= 2U) {
```

```
01128                 uint16_t totalCertLen = (static_cast<uint16_t>(response[0]) « 8U)
01129                                         | static_cast<uint16_t>(response[1]);
01130
01131                 if (totalCertLen <= CW_MANUF_CERT_MAX_BYTES) {
01132                     uint16_t certInPage = static_cast<uint16_t>(dataBytes - 2U);
01133                     if (certInPage > totalCertLen) {
01134                         certInPage = totalCertLen;
01135                     }
01136                     (void)CW_Utils::safe_memcpy(cert, CW_MANUF_CERT_MAX_BYTES, response + 2U,
01137                                                 static_cast<size_t>(certInPage));
01138                     certLen = certInPage;
01139
01140                     uint8_t pageIdx = 1U;
01141                     while ((certLen < totalCertLen) && (pageIdx < 8U)) {
01142                         apdu[APDU_P2_IDX] = pageIdx;
01143                         responseLen = static_cast<uint16_t>(sizeof(response));
01144
01145                         if (!_driver.sendAPDULarge(apdu, static_cast<uint8_t>(sizeof(apdu)),
01146                                                    response, responseLen)) {
01147                             break;
01148                         }
01149                         if (responseLen > static_cast<uint16_t>(sizeof(response))) {
01150 #if CW_DEBUG_LOGGING
01151                             _logger.println(F("getManufacturerCertificate: driver reported
      overflow."));
01152 #endif
01153                             return false;
01154                         }
01155                         if (!checkStatusWord(response, responseLen, 0x90U, 0x00U)) {
01156                             break;
01157                         }
01158
01159                         uint16_t pageData = static_cast<uint16_t>(
01160                             responseLen - static_cast<uint16_t>(RESPONSE_STATUS_WORDS_IN_BYTES));
01161                         uint16_t remaining = static_cast<uint16_t>(totalCertLen - certLen);
01162                         if (pageData > remaining) {
01163                             pageData = remaining;
01164                         }
01165
01166                         if ((certLen + pageData) > static_cast<uint16_t>(CW_MANUF_CERT_MAX_BYTES)) {
01167                             break;
01168                         }
01169                         (void)CW_Utils::safe_memcpy(cert + certLen,
01170                                                     CW_MANUF_CERT_MAX_BYTES -
      static_cast<size_t>(certLen),
01171                                                     response, static_cast<size_t>(pageData));
01172                         certLen = static_cast<uint16_t>(certLen + pageData);
01173                         pageIdx++;
01174                     }
01175
01176                     ret = (certLen == totalCertLen);
01177                     if (!ret) {
01178 #if CW_DEBUG_LOGGING
01179                         _logger.println(F("getManufacturerCertificate: incomplete."));
01180 #endif
01181                     }
01182                 } else {
01183 #if CW_DEBUG_LOGGING
01184                     _logger.println(F("getManufacturerCertificate: cert too large."));
01185 #endif
01186                 }
01187             }
01188         }
01189     }
01190
01191     return ret;
01192 }
01193
01194 bool CW_SecureChannel::preFetchManufacturerCert() {
01195     _cachedMfCertLen = 0U;
01196     bool result = getManufacturerCertificate(s_mfCertBuf, _cachedMfCertLen);
01197     if (!result) {
01198         _cachedMfCertLen = 0U;
01199     }
01200     return result;
01201 }
01202
01227 uint8_t CW_SecureChannel::verifyCertificateChain(const uint8_t* cardCert,
01228                                                  uint8_t cardCertLen) {
01229     uint8_t result = CW_CERT_OK;
01230
01231     if ((cardCert == NULL) || (cardCertLen < 80U) || (cardCert[0] != 0x43U)) {
01232 #if CW_DEBUG_LOGGING
01233         _logger.println(F("verifyCert: invalid card cert."));
01234 #endif
01235         result = CW_CERT_FORMAT_ERROR;
01236     }
```

```
01237
01238     /* Consume the pre-fetched cert (filled by preFetchManufacturerCert() before
01239      * getCardCertificate() was called).  Fall back to fetching now if none cached --
01240      * this will fail on cards whose state machine has already advanced past INS=F7. */
01241     uint16_t mfCertLen = _cachedMfCertLen;
01242     _cachedMfCertLen = 0U;
01243
01244     if (result == CW_CERT_OK) {
01245         if (mfCertLen == 0U) {
01246             if (!getManufacturerCertificate(s_mfCertBuf, mfCertLen) || (mfCertLen < 20U)) {
01247 #if CW_DEBUG_LOGGING
01248                 _logger.println(F("verifyCert: failed to get mfr cert."));
01249 #endif
01250                 result = CW_CERT_FORMAT_ERROR;
01251             }
01252         } else if (mfCertLen < 20U) {
01253 #if CW_DEBUG_LOGGING
01254             _logger.println(F("verifyCert: pre-fetched mfr cert too short."));
01255 #endif
01256             result = CW_CERT_FORMAT_ERROR;
01257         } else {
01258             /* Pre-fetched cert in s_mfCertBuf is valid -- no APDU needed. */
01259         }
01260     }
01261
01262     /* MED-05: replace byte-pattern search with a structural DER walker that
01263      * enforces field order.  This prevents attacker-planted OID sequences in
01264      * unsigned extensions from being picked up as the device public key. */
01265     uint16_t       tbsMsgStart = 0U;
01266     uint16_t       tbsMsgLen   = 0U;
01267     const uint8_t* pubKey65Ptr = NULL;
01268     const uint8_t* mfSigPtr    = NULL;
01269     uint8_t        mfSigLen    = 0U;
01270
01271     if (result == CW_CERT_OK) {
01272         if (!derWalkMfCert(s_mfCertBuf, mfCertLen,
01273                            tbsMsgStart, tbsMsgLen,
01274                            pubKey65Ptr,
01275                            mfSigPtr, mfSigLen)) {
01276 #if CW_DEBUG_LOGGING
01277             _logger.println(F("verifyCert: DER walk of mfr cert failed."));
01278 #endif
01279             result = CW_CERT_KEY_NOT_FOUND;
01280         }
01281     }
01282
01283     const uint8_t* devicePubKey64 = NULL;
01284     if (result == CW_CERT_OK) {
01285         devicePubKey64 = pubKey65Ptr + 1U; /* skip the 0x04 uncompressed prefix */
01286
01287         const uint8_t* mfMsg    = s_mfCertBuf + tbsMsgStart;
01288         uint16_t       mfMsgLen = tbsMsgLen;
01289
01290         bool mfVerified = false;
01291         for (uint8_t i = 0U; i < CW_TRUSTED_CA_COUNT; i++) {
01292             if (verifyEcdsaSha256(CW_TRUSTED_CA_KEYS[i],
01293                                   mfMsg, mfMsgLen,
01294                                   mfSigPtr, mfSigLen)) {
01295                 mfVerified = true;
01296                 break;
01297             }
01298         }
01299         if (!mfVerified) {
01300 #if CW_DEBUG_LOGGING
01301             _logger.println(F("verifyCert: mfr cert sig INVALID -- card NOT genuine."));
01302 #endif
01303             result = CW_CERT_MANUF_SIG_INVALID;
01304         }
01305 #if CW_DEBUG_LOGGING
01306         else {
01307             _logger.println(F("Manufacturer cert signature OK."));
01308         }
01309 #endif
01310     }
01311
01312     const uint8_t CARD_CERT_MSG_LEN = 74U;
01313     if (result == CW_CERT_OK) {
01314         if (cardCertLen <= CARD_CERT_MSG_LEN) {
01315 #if CW_DEBUG_LOGGING
01316             _logger.println(F("verifyCert: card cert too short for sig."));
01317 #endif
01318             result = CW_CERT_FORMAT_ERROR;
01319         }
01320         else {
01321             const uint8_t* cardSig    = cardCert + CARD_CERT_MSG_LEN;
01322             uint8_t        cardSigLen = cardCertLen - CARD_CERT_MSG_LEN;
01323
```

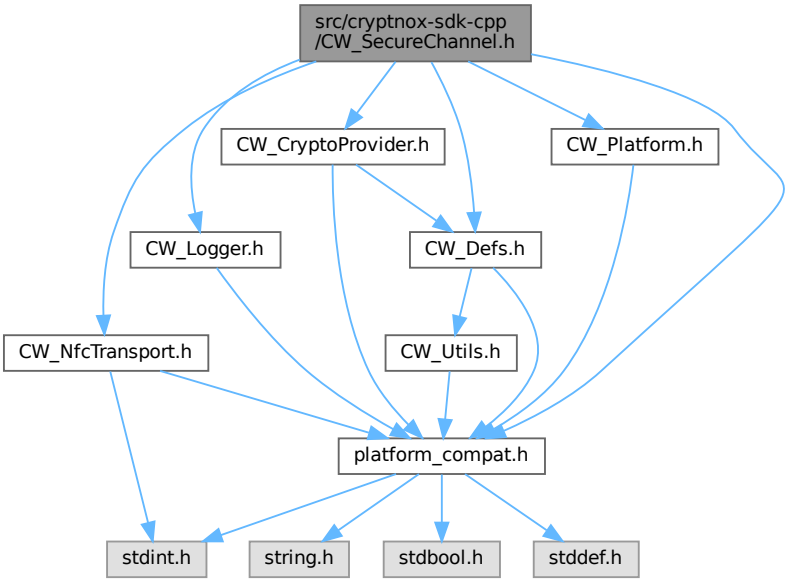
```
01324             if (!verifyEcdsaSha256(devicePubKey64,
01325                                    cardCert, CARD_CERT_MSG_LEN,
01326                                    cardSig, cardSigLen)) {
01327 #if CW_DEBUG_LOGGING
01328                 _logger.println(F("verifyCert: card cert sig INVALID."));
01329 #endif
01330                 result = CW_CERT_CARD_SIG_INVALID;
01331             }
01332 #if CW_DEBUG_LOGGING
01333             else {
01334                 _logger.println(F("Card cert signature OK."));
01335             }
01336 #endif
01337         }
01338     }
01339
01340     if (result == CW_CERT_OK) {
01341         if ((cardCertLen <= (1U + CW_CERT_NONCE_SIZE)) ||
01342             (!CW_Utils::secure_compare(cardCert + 1U, _lastNonce, CW_CERT_NONCE_SIZE))) { /* M-03:
      constant-time nonce compare */
01343 #if CW_DEBUG_LOGGING
01344             _logger.println(F("verifyCert: nonce mismatch -- possible replay."));
01345 #endif
01346             result = CW_CERT_NONCE_MISMATCH;
01347         }
01348     }
01349
01350 #if CW_DEBUG_LOGGING
01351     if (result == CW_CERT_OK) {
01352         _logger.println(F("Certificate chain OK. Card is genuine."));
01353     }
01354 #endif
01355
01356     /* Wipe manufacturer cert buffer -- it held the card device public key (M-01). */
01357     CW_Utils::secure_wipe(s_mfCertBuf, sizeof(s_mfCertBuf));
01358
01359     return result;
01360 }
```

6.47 src/cryptnox-sdk-cpp/CW_SecureChannel.h File Reference

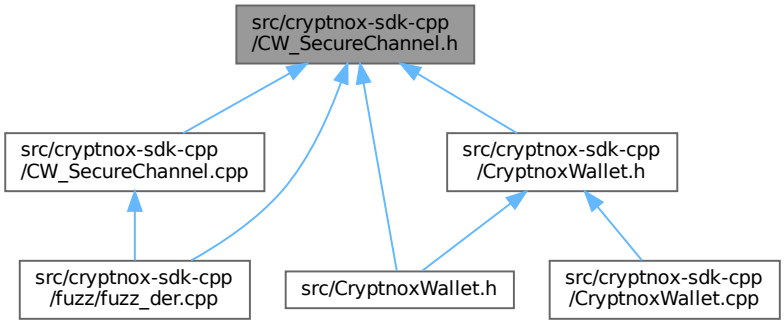
Cryptnox secure channel protocol over NFC.

```
#include "platform_compat.h"
#include "CW_NfcTransport.h"
#include "CW_Logger.h"
#include "CW_CryptoProvider.h"
#include "CW_Platform.h"
#include "CW_Defs.h"
```

Include dependency graph for CW_SecureChannel.h:



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



Classes

- class CW_SecureChannel

  Implements the Cryptnox secure channel protocol over NFC.

Macros

- #define CW_PAIRING_DATA "Cryptnox Basic CommonPairingData"
- #define CW_PAIRING_DATA_BYTES (sizeof(CW_PAIRING_DATA) - 1U)

6.47.1 Detailed Description

Cryptnox secure channel protocol over NFC.

Declares CW_SecureChannel, responsible for every low-level APDU exchange required to establish and use an authenticated, encrypted session with a Cryptnox Hardware Wallet:

- SELECT AID
- Manufacturer + card certificate retrieval and chain verification
- Ephemeral key extraction and ECDH session key derivation
- MUTUALLY AUTHENTICATE
- AES-CBC + MAC secure messaging (encrypt / decrypt / SW check)

The class is composed inside CryptnoxWallet and is not normally used directly by application code. Definition in file CW_SecureChannel.h.

6.47.2 Macro Definition Documentation

6.47.2.1 CW_PAIRING_DATA

```
#define CW_PAIRING_DATA "Cryptnox Basic CommonPairingData"
```

Definition at line 30 of file CW_SecureChannel.h.

6.47.2.2 CW_PAIRING_DATA_BYTES

```
#define CW_PAIRING_DATA_BYTES (sizeof(CW_PAIRING_DATA) - 1U)
```

Definition at line 31 of file CW_SecureChannel.h.

6.48 CW_SecureChannel.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
00022
00023 #ifndef CW_SECURECHANNEL_H
00024 #define CW_SECURECHANNEL_H
00025
00026 /**************************************************************
00027  * 1. Public constants
00028  **************************************************************/
00029
00030 #define CW_PAIRING_DATA       "Cryptnox Basic CommonPairingData"
00031 #define CW_PAIRING_DATA_BYTES (sizeof(CW_PAIRING_DATA) - 1U)
00032
00033 /**************************************************************
00034  * 2. Included files
00035  **************************************************************/
00036
00037 #include "platform_compat.h"
00038 #include "CW_NfcTransport.h"
00039 #include "CW_Logger.h"
00040 #include "CW_CryptoProvider.h"
00041 #include "CW_Platform.h"
00042 #include "CW_Defs.h"          /* for CW_SecureSession, CW_Curve, and constants */
00043
00044 /**************************************************************
00045  * 2. Class declaration
00046  **************************************************************/
00047
00064 class CW_SecureChannel {
00065 public:
00074     CW_SecureChannel(CW_NfcTransport& driver, CW_Logger& logger,
00075                      CW_CryptoProvider& crypto, CW_Platform& platform);
00076
00077     CW_SecureChannel(const CW_SecureChannel&) = delete;
00078     CW_SecureChannel& operator=(const CW_SecureChannel&) = delete;
00079
00084     bool begin();
00085
00090     bool inListPassiveTarget();
00091
```

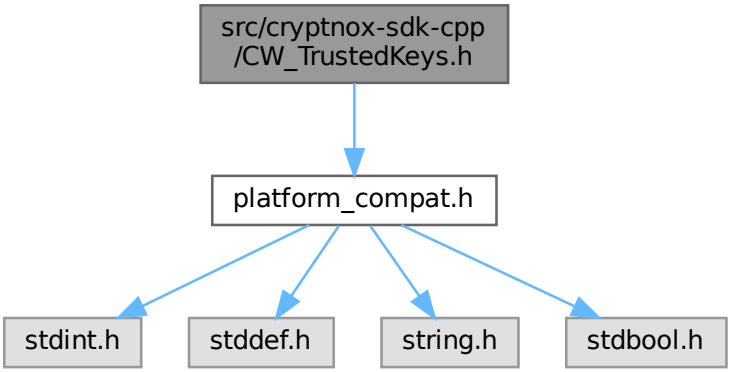
```
00095     void resetReader();
00096
00101     bool printFirmwareVersion();
00102
00107     bool selectApdu();
00108
00120     bool getCardCertificate(uint8_t* cardCertificate, uint8_t& cardCertificateLength);
00121
00130     bool extractCardEphemeralKey(const uint8_t* cardCertificate,
00131                                  uint8_t* cardEphemeralPubKey,
00132                                  uint8_t* fullEphemeralPubKey65 = NULL);
00133
00151     bool openSecureChannel(uint8_t* salt,
00152                            uint8_t* clientPublicKey,
00153                            uint8_t* clientPrivateKey,
00154                            CW_Curve sessionCurve);
00155
00183     bool mutuallyAuthenticate(CW_SecureSession& session,
00184                               const uint8_t* salt,
00185                               uint8_t* clientPublicKey,
00186                               const uint8_t* clientPrivateKey,
00187                               CW_Curve sessionCurve,
00188                               const uint8_t* cardEphemeralPubKey);
00189
00197     bool getManufacturerCertificate(uint8_t* cert, uint16_t& certLen);
00198
00209     bool preFetchManufacturerCert();
00210
00238     uint8_t verifyCertificateChain(const uint8_t* cardCert, uint8_t cardCertLen);
00239
00269     bool aesCbcEncrypt(CW_SecureSession& session,
00270                        const uint8_t apdu[], uint16_t apduLength,
00271                        const uint8_t data[], uint16_t dataLength,
00272                        uint8_t* decryptedOutput = NULL,
00273                        uint16_t* decryptedOutputLength = NULL);
00274
00294     bool aesCbcDecrypt(const CW_SecureSession& session,
00295                        uint8_t* response, size_t responseLen,
00296                        uint8_t* macValue,
00297                        uint8_t* decryptedOutput = NULL,
00298                        uint16_t* decryptedOutputLength = NULL);
00299
00309     bool checkStatusWord(const uint8_t* response, uint16_t responseLength,
00310                          uint8_t sw1Expected, uint8_t sw2Expected);
00311
00312 private:
00313     CW_NfcTransport&   _driver;
00314     CW_Logger&         _logger;
00315     CW_CryptoProvider& _crypto;
00316     CW_Platform&       _platform;
00317
00319     uint8_t _lastNonce[CW_CERT_NONCE_SIZE];
00320
00322     uint16_t _cachedMfCertLen;
00323
00324     static bool parseDerSigToRaw(const uint8_t* der, uint8_t derLen,
00325                                  uint8_t* raw64);
00326
00327     bool verifyEcdsaSha256(const uint8_t* pubKey64,
00328                            const uint8_t* message, uint16_t msgLen,
00329                            const uint8_t* derSig, uint8_t derSigLen);
00330
00331 #ifdef CW_FUZZ_BUILD
00332     friend struct DerFuzzTarget;
00333 #endif
00334 };
00335
00336 #endif // CW_SECURECHANNEL_H
```

6.49 src/cryptnox-sdk-cpp/CW_TrustedKeys.h File Reference

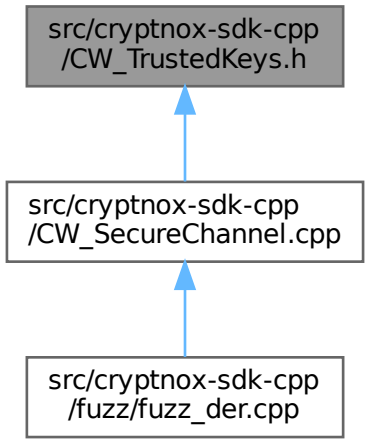
Cryptnox CA public keys used for offline certificate verification.

```
#include "platform_compat.h"
```

Include dependency graph for CW_TrustedKeys.h:



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



Macros

- #define CW_TRUSTED_CA_COUNT (1U)

Variables

- static const uint8_t CW_CA_DLT_PUBKEY [64]
- static const uint8_t *const CW_TRUSTED_CA_KEYS [CW_TRUSTED_CA_COUNT]

6.49.1 Detailed Description

Cryptnox CA public keys used for offline certificate verification.

Holds the trusted secp256r1 public keys that CW_SecureChannel::verifyCertificateChain checks the manufacturer certificate against. Each key is stored as 64 raw bytes (X[32] || Y[32], no 0x04 prefix) so it can be passed directly to `uECC_verify()`.
Source: https://verify.cryptnox.tech/certificates/
To update a key: download the .crt file from the URL above and extract the EC public-key coordinates with:

```
python -c "from cryptography import x509; \
  c=x509.load_pem_x509_certificate(open('CERT.crt','rb').read()); \
  n=c.public_key().public_numbers(); \
  print(hex(n.x)); print(hex(n.y))"
```

Definition in file CW_TrustedKeys.h.

6.49.2 Macro Definition Documentation

6.49.2.1 CW_TRUSTED_CA_COUNT

```
#define CW_TRUSTED_CA_COUNT (1U)
```

Definition at line 46 of file CW_TrustedKeys.h.
Referenced by CW_SecureChannel::verifyCertificateChain().

6.49.3 Variable Documentation

6.49.3.1 CW_CA_DLT_PUBKEY

```
const uint8_t CW_CA_DLT_PUBKEY[64]  [static]
```

Initial value:

```
= {

    0x64, 0x7a, 0x11, 0x2c, 0x2a, 0xcf, 0x5a, 0x49,
    0xb5, 0x2b, 0x32, 0x1c, 0xbd, 0xcf, 0x5a, 0x9e,
    0xa3, 0x07, 0xfc, 0x4c, 0xcd, 0x33, 0xaa, 0x78,
    0xb9, 0xb8, 0x05, 0x5d, 0xd8, 0x4d, 0x03, 0xae,

    0xda, 0xb0, 0x2c, 0xc7, 0x20, 0xa7, 0x80, 0xcb,
    0x1f, 0xf2, 0x80, 0xaf, 0x50, 0x77, 0x4a, 0x6c,
    0xdc, 0x15, 0x7e, 0xfa, 0x23, 0x5e, 0xa2, 0x53,
    0x11, 0xa4, 0x2b, 0x4c, 0xf5, 0x7d, 0x88, 0x61
}
```

Definition at line 32 of file CW_TrustedKeys.h.

6.49.3.2 CW_TRUSTED_CA_KEYS

```
const uint8_t* const CW_TRUSTED_CA_KEYS[CW_TRUSTED_CA_COUNT]  [static]
```

Initial value:

```
= {
    CW_CA_DLT_PUBKEY
}
```

Definition at line 50 of file CW_TrustedKeys.h.
Referenced by CW_SecureChannel::verifyCertificateChain().

6.50 CW_TrustedKeys.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
00026
00027 #ifndef CW_TRUSTEDKEYS_H
00028 #define CW_TRUSTEDKEYS_H
00029
00030 #include "platform_compat.h"
00031
00032 /* CRYPTNOX_DLT_CARDS_CA -- signs manufacturer certificates for DLT cards.
00033  * Curve: secp256r1. Issued by CRYPTNOX INTERMEDIATE CA #2. */
00034 static const uint8_t CW_CA_DLT_PUBKEY[64] = {
```

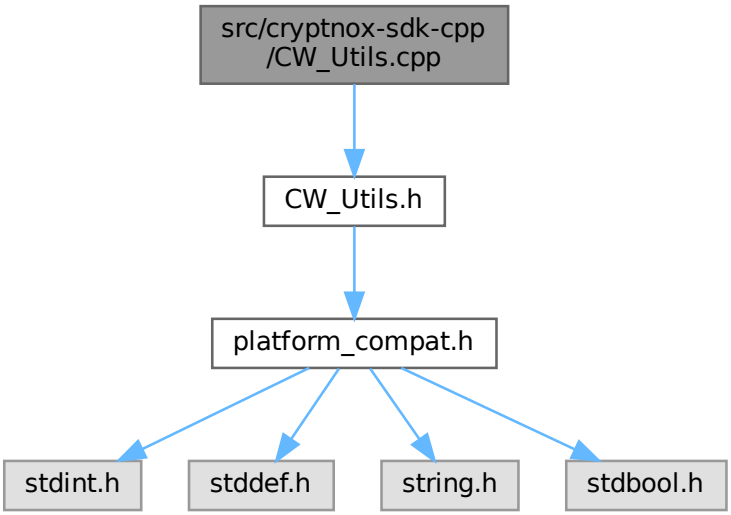
```
00035     /* X */
00036     0x64, 0x7a, 0x11, 0x2c, 0x2a, 0xcf, 0x5a, 0x49,
00037     0xb5, 0x2b, 0x32, 0x1c, 0xbd, 0xcf, 0x5a, 0x9e,
00038     0xa3, 0x07, 0xfc, 0x4c, 0xcd, 0x33, 0xaa, 0x78,
00039     0xb9, 0xb8, 0x05, 0x5d, 0xd8, 0x4d, 0x03, 0xae,
00040     /* Y */
00041     0xda, 0xb0, 0x2c, 0xc7, 0x20, 0xa7, 0x80, 0xcb,
00042     0x1f, 0xf2, 0x80, 0xaf, 0x50, 0x77, 0x4a, 0x6c,
00043     0xdc, 0x15, 0x7e, 0xfa, 0x23, 0x5e, 0xa2, 0x53,
00044     0x11, 0xa4, 0x2b, 0x4c, 0xf5, 0x7d, 0x88, 0x61
00045 };
00046
00047 /* Number of trusted CA keys in the table below. */
00048 #define CW_TRUSTED_CA_COUNT (1U)
00049
00050 /* Table of all trusted CA public keys (each 64 bytes, X||Y).
00051  * verifyCertificateChain() tries every entry until one succeeds. */
00052 static const uint8_t* const CW_TRUSTED_CA_KEYS[CW_TRUSTED_CA_COUNT] = {
00053     CW_CA_DLT_PUBKEY
00054 };
00055
00056 #endif /* CW_TRUSTEDKEYS_H */
```

6.51 src/cryptnox-sdk-cpp/CW_Utils.cpp File Reference

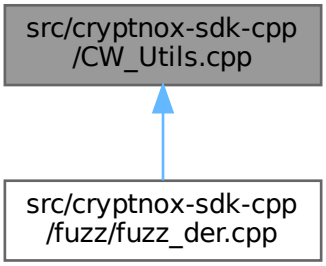
Implementation of the platform-independent utility helpers.

`#include "CW_Utils.h"`

Include dependency graph for CW_Utils.cpp:



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



6.51.1 Detailed Description

Implementation of the platform-independent utility helpers.
CW_Utils::secure_compare iterates over the full length to avoid leaking byte-position information through timing. CW_Utils::secure_wipe uses a volatile pointer so the writes cannot be optimised away. CW_Utils::safe_memcpy validates pointers, length, and source/destination overlap before delegating to memcpy.
Definition in file CW_Utils.cpp.

6.52 CW_Utils.cpp

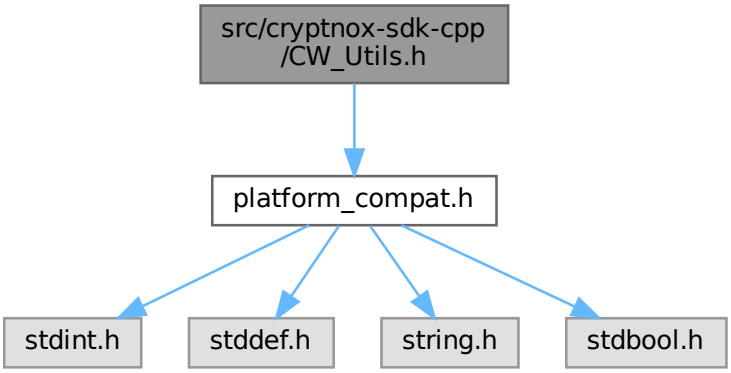
Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00016
00017 #include "CW_Utils.h"
00018
00022 bool CW_Utils::secure_compare(const uint8_t* a, const uint8_t* b, size_t len) {
00023     bool ret = false;
00024     if ((a != NULL) && (b != NULL) && (len > 0U)) {
00025         uint8_t diff = 0U;
00026         for (size_t i = 0U; i < len; i++) {
00027             diff |= a[i] ^ b[i];
00028         }
00029         ret = (diff == 0U);
00030     }
00031     return ret;
00032 }
00033
00037 void CW_Utils::secure_wipe(uint8_t* buf, size_t len) {
00038     if ((buf != NULL) && (len > 0U)) {
00039         volatile uint8_t* p = buf;
00040         for (size_t i = 0U; i < len; i++) {
00041             p[i] = 0U;
00042         }
00043     }
00044 }
00045
00050 bool CW_Utils::safe_memcpy(uint8_t* dst, size_t dstSize,
00051                                 const uint8_t* src, size_t count) {
00052     bool ret = false;
00053     if ((dst != NULL) && (src != NULL) && (count > 0U) && (count <= dstSize)) {
00054         bool overlap = (dst < (src + count)) && (src < (dst + dstSize));
00055         if (!overlap) {
00056             memcpy(dst, src, count);
00057             ret = true;
00058         }
00059     }
```

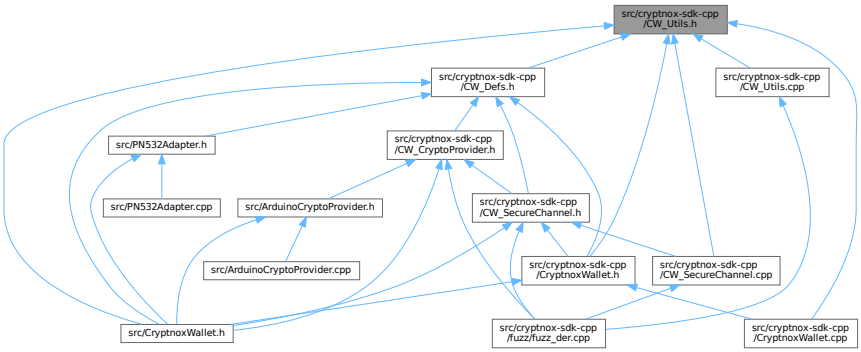
```
00060     return ret;
00061 }
```

6.53 src/cryptnox-sdk-cpp/CW_Utils.h File Reference

Platform-independent security and memory utilities.
`#include "platform_compat.h"`
Include dependency graph for CW_Utils.h:



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



Classes

- class CW_Utils

  Portable utility functions for cryptographic and security operations.

6.53.1 Detailed Description

Platform-independent security and memory utilities.
Declares CW_Utils, a small collection of helpers used throughout the SDK that have no platform dependencies:

- constant-time buffer comparison (timing-side-channel safe)

- guaranteed-not-elided secure wipe of sensitive buffers
- bounds-checked, overlap-checked memcpy
- secure random byte generation (delegated to a platform impl)

Hardware-specific helpers (e.g. TRNG bring-up) live in the concrete crypto provider rather than here.
Definition in file CW_Utils.h.

6.54 CW_Utils.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
00020
00021 #ifndef CW_UTILS_H
00022 #define CW_UTILS_H
00023
00024 /**************************************************************
00025  * 1. Included files
00026  **************************************************************/
00027
00028 #include "platform_compat.h"
00029
00030 /**************************************************************
00031  * 2. Class declaration
00032  **************************************************************/
00033
00045 class CW_Utils {
00046 public:
00059     static bool secure_compare(const uint8_t* a, const uint8_t* b, size_t len);
00060
00070     static void secure_wipe(uint8_t* buf, size_t len);
00071
00081     static bool safe_memcpy(uint8_t* dst, size_t dstSize,
00082                             const uint8_t* src, size_t count);
00083
00095     static bool fill_secure_random(uint8_t* dest, size_t len);
00096 };
00097
00098 #endif // CW_UTILS_H
```

6.55 src/cryptnox-sdk-cpp/docs/patch_latex.py File Reference

Namespaces

- namespace patch_latex

Functions

- patch_latex._to_ascii (match)

Variables

- patch_latex.path = sys.argv[1]
- int patch_latex.title = sys.argv[2] if len(sys.argv) > 2 else None
- patch_latex.url
- patch_latex.write_to
- patch_latex.encoding
- patch_latex.s = _f.read()
- patch_latex.sty = os.path.join(os.path.dirname(path), "doxygen.sty")
- patch_latex.d = _f.read()
- patch_latex.idx = os.path.join(os.path.dirname(path), "index.tex")
- patch_latex.i = _f.read()

- patch_latex.keep
- patch_latex.tex
- patch_latex.body = _f.read()
- str patch_latex.marker = "\\begin{document}"
- patch_latex.split = body.find(marker)
- int patch_latex.head = -1 else ""
- int patch_latex.tail = -1 else body
- int patch_latex.fixed = head + re.sub(r"[^\x00-\x7F]", _to_ascii, tail)

6.56 patch_latex.py

Go to the documentation of this file.

```
00001 """Post-process Doxygen's refman.tex before pdflatex, to match the Cryptnox PDF style.
00002
00003 Usage: python3 docs/patch_latex.py docs/latex/refman.tex ["Friendly Title"]
00004 """
00005 import os
00006 import re
00007 import sys
00008
00009 path = sys.argv[1]
00010 title = sys.argv[2] if len(sys.argv) > 2 else None
00011
00012 # pdflatex can't embed the committed SVG logo directly, so convert it to a PDF beside
00013 # refman.tex for the \includegraphics{cryptnox-logo-dark} on the cover (cairosvg in CI).
00014 try:
00015     import cairosvg
00016 except ImportError as _e:
00017     raise RuntimeError(
00018         "PDF docs build requires cairosvg to generate the cover logo; "
00019         "install it (pip install cairosvg)."
00020     ) from _e
00021 cairosvg.svg2pdf(
00022     url=os.path.join(os.path.dirname(path), os.pardir, "cryptnox-logo.svg"),
00023     write_to=os.path.join(os.path.dirname(path), "cryptnox-logo-dark.pdf"),
00024 )
00025
00026 with open(path, encoding="utf-8") as _f:
00027     s = _f.read()
00028
00029 # Use TeX Gyre Heros (sans) for the whole document, like the Sphinx PDFs.
00030 # Doxygen already defaults to \sfdefault; just swap its Helvetica for TeX Gyre Heros.
00031 s = s.replace(r"\usepackage[scaled=.90]{helvet}", "\\usepackage{tgheros}\n\\usepackage{tgtermes}")
00032
00033 # Remove the "Generated by Doxygen X.Y.Z" credit from the title page
00034 s = re.sub(r"\{\\large Generated by Doxygen[^}]*\}\\\\\s*", "", s)
00035
00036 # Right-align the title-page block and put the logo inside the same flushright as the
00037 # title, with identical spacing to the Sphinx covers (\vskip 1em / logo / \vskip 2em).
00038 # Must run BEFORE the title-font edit below, which rewrites \Large -> \LARGE\bfseries.
00039 s = s.replace(
00040     "\\begin{center}%\n  {\\Large",
00041     "\\begin{flushright}%\n"
00042     "  \\vskip -0.5em%\n"
00043     "  \\includegraphics[width=7cm]{cryptnox-logo-dark}\\par\n"
00044     "  \\vskip 2em%\n"
00045     "  {\\Large",
00046 )
00047 s = s.replace(
00048     "\\end{center}\n  \\end{titlepage}",
00049     "\\end{flushright}\n  \\vfill\n  \\begin{flushright}\\large June 20, 2026\\end{flushright}\n
    \\end{titlepage}",
00050 )
00051
00052 # Title: use the friendly name (passed as arg) + "Manual", in the Sphinx cover font.
00053 # Falls back to the Doxygen project name if no title is given.
00054 if title:
00055     s = re.sub(r"\{\\Large [^}]*\}\\\\",
00056                lambda m: r"{\LARGE\sffamily\bfseries " + title + r" Manual}\\", s, count=1)
00057 else:
00058     s = re.sub(r"\{\\Large ([^}]*)\}\\\\", r"{\\LARGE\\sffamily\\bfseries \1 Manual}\\\\", s, count=1)
00059
00060 # Version: prefix with "Release", unify the font (\large\itshape), and add more space
00061 # above it (title -> version gap), via the \\ optional length
00062 s = re.sub(r"\[1ex\]\\large (.*?) \\\\", r"[1.5em]{\\large\\itshape Release \1}\\\\", s, count=1)
00063
00064 # Headings (chapters, sections, "Contents") in the sans heading font, like the
00065 # Sphinx PDFs and the Yubico manual (TeX Gyre Heros bold), keeping Doxygen's gray.
00066 s = re.sub(
```

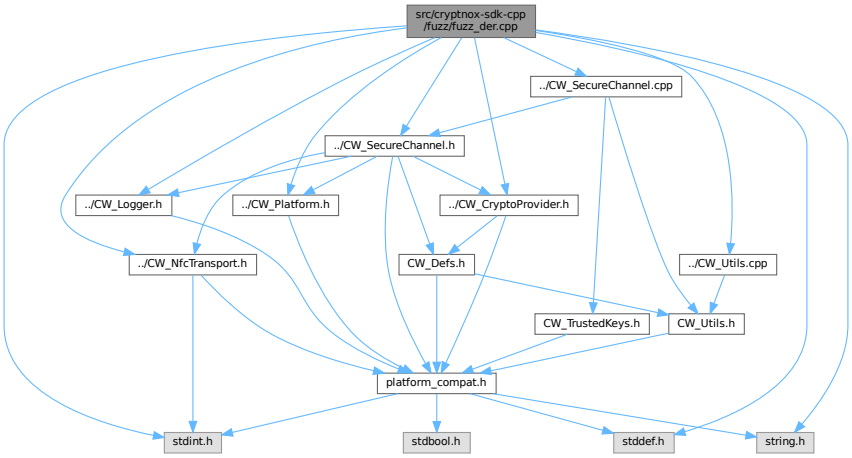
```
00067     r"\\doxyallsectionsfont\{%\s*\\fontseries\{bc\}\\selectfont%\s*\\color\{darkgray\}%\s*\}",
00068     lambda m: (
00069         r"\doxyallsectionsfont{\sffamily\bfseries\color{black}}"
00070         "\n\\makeatletter"
00071         "\n\\renewcommand{\\@makechapterhead}[1]{%"
00072         "\n  \\vspace*{50\\p@}{\\parindent\\z@\\raggedright\\sffamily"
00073         "\n    \\ifnum\\c@secnumdepth>\\m@ne"
00074         "\n      \\huge\\bfseries\\color{black}\\@chapapp\\space\\thechapter\\par\\nobreak\\vskip
    20\\p@\\fi"
00075         "\n    \\interlinepenalty\\@M\\Huge\\bfseries\\color{black}#1\\par\\nobreak\\vskip 40\\p@}}"
00076         "\n\\renewcommand{\\@makeschapterhead}[1]{%"
00077         "\n  \\vspace*{50\\p@}{\\parindent\\z@\\raggedright\\sffamily"
00078         "\n    \\interlinepenalty\\@M\\Huge\\bfseries\\color{black}#1\\par\\nobreak\\vskip 40\\p@}}"
00079         "\n\\makeatother"
00080     ),
00081     s, count=1,
00082 )
00083
00084 # One-sided layout (no blank filler pages)
00085 s = s.replace(r"\documentclass[twoside]", r"\documentclass[oneside]")
00086
00087 # Footer: replace "Generated by Doxygen" with the copyright, in the sans heading font
00088 # (footer only; the title-page credit uses \large, so it is left untouched)
00089 s = s.replace(
00090     r"\bfseries\scriptsize Generated by Doxygen",
00091     r"\sffamily\fontsize{11}{13.6}\selectfont \copyright{} 2026 Cryptnox SA",
00092 )
00093
00094 # Running header: sans, regular weight, 11pt -- same size as the Sphinx PDFs
00095 s = s.replace(r"\fancyhead[LE, RO]{\bfseries\thepage}", r"\fancyhead[LE,
    RO]{\sffamily\fontsize{11}{13.6}\selectfont\thepage}")
00096 s = s.replace(r"\fancyhead[LO]{\bfseries\rightmark}",
    r"\fancyhead[LO]{\sffamily\fontsize{11}{13.6}\selectfont\rightmark}")
00097 s = s.replace(r"\fancyhead[RE]{\bfseries\leftmark}",
    r"\fancyhead[RE]{\sffamily\fontsize{11}{13.6}\selectfont\leftmark}")
00098
00099 # Sans-serif TOC entries
00100 s = s.replace(r"\tableofcontents", r"\addtocontents{toc}{\protect\sffamily}\tableofcontents", 1)
00101
00102 # 4pt gap between header/footer text and their rules
00103 s = s.replace(
00104     r"\pagestyle{fancyplain}",
00105     r"\def\headruleskip{4pt}\def\footruleskip{4pt}\pagestyle{fancyplain}",
00106     1,
00107 )
00108
00109 # pdflatex (inputenc utf8) aborts on any Unicode char it doesn't know. Declare the
00110 # math relations we want rendered as real glyphs (used in @c doc comments, e.g.
00111 # "len <= 254"); everything else non-ASCII is transliterated to safe ASCII at the end
00112 # of this script. Without this, pdflatex stops with "Unicode character not set up".
00113 s = s.replace(
00114     r"\begin{document}",
00115     "\\DeclareUnicodeCharacter{2264}{\\ensuremath{\\leq}}\n"
00116     "\\DeclareUnicodeCharacter{2265}{\\ensuremath{\\geq}}\n"
00117     "\\begin{document}",
00118     1,
00119 )
00120
00121 # Drop the alphabetical index
00122 s = s.replace(r"\printindex", "")
00123 s = s.replace(r"\addcontentsline{toc}{chapter}{\indexname}", "")
00124
00125 # Replace the default top gap with just the top rule (the logo is injected inside the
00126 # flushright above, matching the Sphinx layout)
00127 s = s.replace(r"\vspace*{7cm}", r"\noindent\rule{\textwidth}{1pt}\par")
00128
00129 with open(path, "w", encoding="utf-8") as _f:
00130     _f.write(s)
00131 print("patched", path)
00132
00133 # Section/subsection headings are defined in doxygen.sty (not refman.tex) with a
00134 # hardcoded \normalfont (serif); switch them to the sans heading font + gray.
00135 sty = os.path.join(os.path.dirname(path), "doxygen.sty")
00136 if os.path.exists(sty):
00137     with open(sty, encoding="utf-8") as _f:
00138         d = _f.read()
00139     d = d.replace(r"{\raggedright\normalfont", r"{\raggedright\sffamily\color{black}")
00140     with open(sty, "w", encoding="utf-8") as _f:
00141         _f.write(d)
00142     print("patched", sty)
00143
00144 # Strip the "Download this documentation as PDF" link from the mainpage (README) --
00145 # pointless inside the PDF itself. Kept in README.md for GitHub/HTML.
00146 idx = os.path.join(os.path.dirname(path), "index.tex")
00147 if os.path.exists(idx):
00148     with open(idx, encoding="utf-8") as _f:
00149         i = _f.read()
```

```
00150     i = re.sub(r"^.*Download this documentation as PDF.*\n", "", i, flags=re.MULTILINE)
00151     with open(idx, "w", encoding="utf-8") as _f:
00152         _f.write(i)
00153     print("patched", idx)
00154
00155 # Final safety net: Doxygen emits non-ASCII from source comments and the README
00156 # (box-drawing art in directory trees / wiring diagrams, arrows, stray symbols) into
00157 # every *.tex file. pdflatex's inputenc fatally stops on any codepoint it doesn't
00158 # know, and which ones appear changes as the docs change -- so instead of declaring
00159 # each one, transliterate to ASCII here. Chars LaTeX already renders well are kept;
00160 # box-drawing is mapped to ASCII; anything else is dropped so the build cannot fail
00161 # on an unexpected glyph. Runs last, over all .tex files (the .sty files are pure
00162 # LaTeX and contain no such literals).
00163 import glob
00164
00165 # Codepoints inputenc/textcomp already render (verified in CI logs) plus the math
00166 # relations declared above -- left untouched so their nicer glyphs survive.
00167 keep = frozenset({
00168     0x00A0, 0x00B2, 0x00B3, 0x00D7,           # nbsp, superscript 2/3, multiplication
00169     0x2013, 0x2014, 0x2026,                   # en/em dash, ellipsis
00170     0x2018, 0x2019, 0x201C, 0x201D,           # curly quotes
00171     0x2190, 0x2192, 0x2194,                   # left/right/both arrows
00172     0x2264, 0x2265,                           # <= and >= (declared in the preamble)
00173 })
00174
00175
00176 def _to_ascii(match):
00177     cp = ord(match.group(0))
00178     if cp in keep:
00179         out = match.group(0)
00180     elif 0x2500 <= cp <= 0x257F:
00181         # Box-drawing block: verticals -> '|', corners/tees -> '+', the rest -> '-'.
00182         if cp in (0x2502, 0x2503, 0x2551):
00183             out = "|"
00184         elif 0x250C <= cp <= 0x254B:
00185             out = "+"
00186         else:
00187             out = "-"
00188     else:
00189         out = ""
00190     return out
00191
00192
00193 for tex in glob.glob(os.path.join(os.path.dirname(path), "*.tex")):
00194     with open(tex, encoding="utf-8") as _f:
00195         body = _f.read()
00196     # refman.tex's preamble holds Doxygen's own \doxynewunicodechar{<char>}{...}
00197     # declarations; transliterating those would empty the first argument and crash.
00198     # Only rewrite the document body. Other .tex files are pure body (no \begin{document}),
00199     # so the marker is absent and the whole file is processed.
00200     marker = "\\begin{document}"
00201     split = body.find(marker)
00202     head = body[:split] if split != -1 else ""
00203     tail = body[split:] if split != -1 else body
00204     fixed = head + re.sub(r"[^\x00-\x7F]", _to_ascii, tail)
00205     if fixed != body:
00206         with open(tex, "w", encoding="utf-8") as _f:
00207             _f.write(fixed)
00208         print("transliterated", tex)
```

6.57 src/cryptnox-sdk-cpp/fuzz/fuzz_der.cpp File Reference

```
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include "../CW_CryptoProvider.h"
#include "../CW_NfcTransport.h"
#include "../CW_Logger.h"
#include "../CW_Platform.h"
#include "../CW_SecureChannel.h"
#include "../CW_Utils.cpp"
#include "../CW_SecureChannel.cpp"
```

Include dependency graph for fuzz_der.cpp:



Classes

- class StubNfc
- class StubLogger
- class StubCrypto
- class StubPlatform
- struct DerFuzzTarget

Macros

- #define CW_FUZZ_BUILD 1

Functions

- int LLVMFuzzerTestOneInput (const uint8_t *data, size_t size)

6.57.1 Macro Definition Documentation

6.57.1.1 CW_FUZZ_BUILD

```
#define CW_FUZZ_BUILD 1
```

Definition at line 35 of file fuzz_der.cpp.

6.57.2 Function Documentation

6.57.2.1 LLVMFuzzerTestOneInput()

```
int LLVMFuzzerTestOneInput (
            const uint8_t * data,
            size_t size)
```

Definition at line 130 of file fuzz_der.cpp.
References derWalkMfCert(), and DerFuzzTarget::parseDerSigToRaw().

6.58 fuzz_der.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 /*
00007  * fuzz_der.cpp -- libFuzzer harness for the two DER parser paths in
00008  *                 CW_SecureChannel (SEC-015).
00009  *
00010  * Targets:
00011  *   derWalkMfCert()                   file-static DER X.509 cert walker
00012  *   CW_SecureChannel::parseDerSigToRaw()  private-static DER sig parser
00013  *
00014  * Build (Linux / macOS, clang required):
00015  *   cd cryptnox-sdk-cpp/fuzz
00016  *   mkdir build && cd build
00017  *   cmake .. -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++
00018  *   make
00019  *
00020  * Run:
00021  *   ./fuzz_der corpus/ -max_len=512 -jobs=4
00022  *
00023  * Corpus seeds:
00024  *   Place valid DER ECDSA signatures and manufacturer certificate blobs
00025  *   inside corpus/.  See corpus/README.md for instructions.
00026  *
00027  * Input format (first byte selects target):
00028  *   0x00 + <bytes>  → parseDerSigToRaw only
00029  *   0x01 + <bytes>  → derWalkMfCert only
00030  *   anything else   → both parsers receive the same payload
00031  */
00032
00033 /* Must be defined before including CW_SecureChannel.h so the friend
00034  * declaration for DerFuzzTarget is compiled in.                        */
00035 #define CW_FUZZ_BUILD 1
00036
00037 #include <stdint.h>
00038 #include <stddef.h>
00039 #include <string.h>
00040
00041 /* - SDK headers ------------------------------------ */
00042 #include "../CW_CryptoProvider.h"
00043 #include "../CW_NfcTransport.h"
00044 #include "../CW_Logger.h"
00045 #include "../CW_Platform.h"
00046 #include "../CW_SecureChannel.h"
00047
00048 /* - Minimal concrete stubs for the three abstract interfaces -------
00049  * Required so CW_SecureChannel.cpp compiles; these methods are never
00050  * reached from the DER-parser call paths exercised by this harness.    */
00051
00052 class StubNfc final : public CW_NfcTransport {
00053 public:
00054     bool begin() override { return false; }
00055     bool inListPassiveTarget() override { return false; }
00056     bool sendAPDU(const uint8_t*, uint8_t,
00057                   uint8_t*, uint8_t&) override { return false; }
00058     void resetReader() override {}
00059     bool printFirmwareVersion() override { return false; }
00060 };
00061
00062 class StubLogger final : public CW_Logger {
00063 public:
00064     bool begin(unsigned long) override { return true; }
00065     void print(const __FlashStringHelper*) override {}
00066     void print(const char*) override {}
00067     void print(char) override {}
00068     void print(uint8_t, int) override {}
00069     void print(uint16_t, int) override {}
00070     void print(uint32_t, int) override {}
00071     void print(int, int) override {}
00072     void println() override {}
00073     void println(const __FlashStringHelper*) override {}
00074     void println(const char*) override {}
00075     void println(char) override {}
00076     void println(uint8_t, int) override {}
00077     void println(uint16_t, int) override {}
00078     void println(uint32_t, int) override {}
00079     void println(int, int) override {}
00080 };
00081
00082 class StubCrypto final : public CW_CryptoProvider {
00083 public:
```

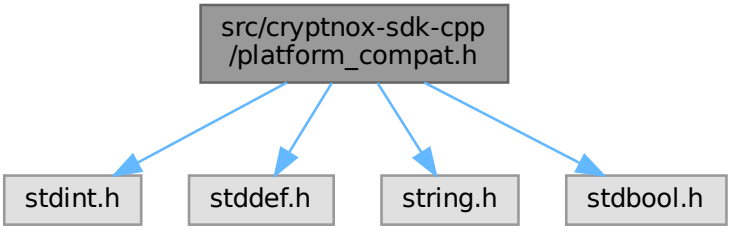
```
00084     bool sha256(const uint8_t*, size_t, uint8_t*) override { return false; }
00085     bool sha512(const uint8_t*, size_t, uint8_t*) override { return false; }
00086     uint16_t aesCbcEncrypt(const uint8_t*, uint16_t, uint8_t*,
00087                            const uint8_t*, uint8_t,
00088                            uint8_t*, bool) override { return 0U; }
00089     uint16_t aesCbcDecrypt(uint8_t*, uint16_t, uint8_t*,
00090                            const uint8_t*, uint8_t,
00091                            uint8_t*, bool) override { return 0U; }
00092     bool ecdh(const uint8_t*, const uint8_t*, uint8_t*,
00093               CW_Curve) override { return false; }
00094     bool makeKey(uint8_t*, uint8_t*,
00095                  CW_Curve) override { return false; }
00096     bool random(uint8_t*, unsigned) override { return false; }
00097     bool ecdsaVerify(const uint8_t*, const uint8_t*, size_t,
00098                      const uint8_t*, CW_Curve) override { return false; }
00099 };
00100
00101 class StubPlatform final : public CW_Platform {
00102 public:
00103     void sleep_ms(uint32_t) override {}
00104 };
00105
00106 /* - CW_Utils::fill_secure_random stub ----------------------
00107  * The real implementation is ESP32-specific (esp32_random.cpp).
00108  * The DER parsers never call this; the stub exists only for the linker. */
00109 bool CW_Utils::fill_secure_random(uint8_t* dest, size_t len) {
00110     if ((dest != NULL) && (len > 0U)) {
00111         memset(dest, 0xA5U, len);
00112     }
00113     return true;
00114 }
00115
00116 /* - Pull in the production code under test ------------------ */
00117 #include "../CW_Utils.cpp"
00118 #include "../CW_SecureChannel.cpp"
00119
00120 /* - DerFuzzTarget: bridge from friend to private parseDerSigToRaw --- */
00121 struct DerFuzzTarget {
00122     static bool parseDerSigToRaw(const uint8_t* der,
00123                                  uint8_t derLen,
00124                                  uint8_t* raw64) {
00125         return CW_SecureChannel::parseDerSigToRaw(der, derLen, raw64);
00126     }
00127 };
00128
00129 /* - libFuzzer entry point ----------------------------- */
00130 extern "C" int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size) {
00131     if (size < 1U) {
00132         return 0;
00133     }
00134
00135     const uint8_t  selector    = data[0];
00136     const uint8_t* payload     = data + 1U;
00137     const size_t   payloadSize = size - 1U;
00138
00139     /* - Target A: parseDerSigToRaw ----------------------- */
00140     if ((selector == 0x00U) || (selector > 0x01U)) {
00141         uint8_t raw64[64U] = { 0U };
00142         uint8_t derLen = (payloadSize <= 255U)
00143                          ? static_cast<uint8_t>(payloadSize)
00144                          : 255U;
00145         (void)DerFuzzTarget::parseDerSigToRaw(payload, derLen, raw64);
00146     }
00147
00148     /* - Target B: derWalkMfCert ------------------------- */
00149     if ((selector == 0x01U) || (selector > 0x01U)) {
00150         uint16_t       tbsMsgStart = 0U;
00151         uint16_t       tbsMsgLen   = 0U;
00152         const uint8_t* pubKey65Ptr = NULL;
00153         const uint8_t* sigPtr      = NULL;
00154         uint8_t        sigLen      = 0U;
00155         uint16_t bufLen = (payloadSize <= 0xFFFFU)
00156                           ? static_cast<uint16_t>(payloadSize)
00157                           : 0xFFFFU;
00158         (void)derWalkMfCert(payload, bufLen,
00159                             tbsMsgStart, tbsMsgLen,
00160                             pubKey65Ptr, sigPtr, sigLen);
00161     }
00162
00163     return 0;
00164 }
```

6.59 src/cryptnox-sdk-cpp/platform_compat.h File Reference

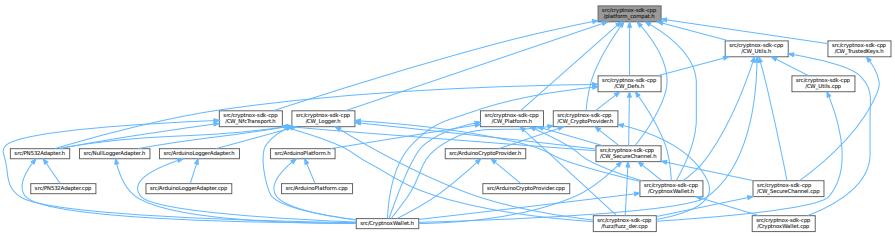
Arduino compatibility shims for non-Arduino (plain C++) builds.

```
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include <stdbool.h>
```

Include dependency graph for platform_compat.h:



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



Classes

- class __FlashStringHelper

Macros

- #define DEC 10
- #define HEX 16
- #define OCT 8
- #define BIN 2
- #define F(string_literal)

6.59.1 Detailed Description

Arduino compatibility shims for non-Arduino (plain C++) builds.
Include this file instead of <Arduino.h> in platform-independent headers. When building for Arduino, ARDUINO is already defined by the toolchain and this file is a no-op — Arduino.h has already provided all these definitions. When building outside Arduino (desktop, ESP-IDF, CI) this file provides the minimum set of defines/types needed so the SDK headers compile cleanly.
Definition in file platform_compat.h.

6.59.2 Macro Definition Documentation

6.59.2.1 BIN

```
#define BIN 2
```

Definition at line 37 of file platform_compat.h.

6.59.2.2 DEC

```
#define DEC 10
```

Definition at line 34 of file platform_compat.h.
Referenced by ArduinoLoggerAdapter::operator=(), CW_Logger::print(), CW_Logger::print(), CW_Logger::print(), CW_Logger::print(), NullLoggerAdapter::print(), NullLoggerAdapter::print(), NullLoggerAdapter::print(), NullLoggerAdapter::print(), CW_Logger::println(), CW_Logger::println(), CW_Logger::println(), CW_Logger::println(), NullLoggerAdapter::println(), NullLoggerAdapter::println(), NullLoggerAdapter::println(), and NullLoggerAdapter::println().

6.59.2.3 F

```
#define F(
              string_literal)
```

Value:

```
(string_literal)
```

Examples

BasicUsage.ino, Connect.ino, Sign.ino, UsdcSigning.ino, and VerifyPin.ino.

Definition at line 42 of file platform_compat.h.
Referenced by CW_SecureChannel::aesCbcDecrypt(), CW_SecureChannel::aesCbcEncrypt(), CW_SecureChannel::checkStatusWord(), CryptnoxWallet::connect(), CryptnoxWallet::debugPrintSignature(), determineYParity(), encodeERC20Transfer(), CryptnoxWallet::establishSecureChannel(), CryptnoxWallet::extractRawSignature(), fetchNonce(), CW_SecureChannel::getCardCertificate(), CryptnoxWallet::getCardInfo(), CW_SecureChannel::getManufacturerCertificate(), loop(), CW_SecureChannel::mutuallyAuthenticate(), CW_SecureChannel::openSecureChannel(), PN532Adapter::printFirmwareVersion(), printHex(), rlpEncodeTxBody(), CW_SecureChannel::selectApdu(), PN532Adapter::sendAPDU(), PN532Adapter::sendAPDULarge(), sendRawTx(), CryptnoxWallet::sendSignApdu(), setup(), CryptnoxWallet::validateSignRequest(), CW_SecureChannel::verifyCertificateChain(), CryptnoxWallet::verifyPin(), and CryptnoxWallet::writeUserData().

6.59.2.4 HEX

```
#define HEX 16
```

Examples

BasicUsage.ino, Sign.ino, and UsdcSigning.ino.

Definition at line 35 of file platform_compat.h.
Referenced by CW_SecureChannel::aesCbcDecrypt(), CW_SecureChannel::aesCbcEncrypt(), CW_SecureChannel::checkStatusWord(), CryptnoxWallet::debugPrintSignature(), CryptnoxWallet::establishSecureChan loop(), PN532Adapter::printFirmwareVersion(), PN532Adapter::sendAPDU(), PN532Adapter::sendAPDULarge(), and setup().

6.59.2.5 OCT

```
#define OCT 8
```

Definition at line 36 of file platform_compat.h.

6.60 platform_compat.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 PLATFORM_COMPAT_H
00007 #define PLATFORM_COMPAT_H
00008
00019
00020 #ifdef ARDUINO
00021 /* On Arduino, pull in Arduino.h so SDK headers that reference __FlashStringHelper,
00022  * DEC/HEX/OCT/BIN, F(), or delay() compile regardless of whether the including
00023  * translation unit already included <Arduino.h>. Arduino.h transitively provides
00024  * <stdint.h>, <stddef.h>, <string.h>, etc., so no need to include them here. */
00025 #  include <Arduino.h>
00026 #else
00027 /* Off Arduino, include the C standard headers explicitly so the SDK compiles
00028  * without depending on what the including TU may or may not have pulled in. */
00029 #  include <stdint.h>
00030 #  include <stddef.h>
00031 #  include <string.h>
00032 #  include <stdbool.h>
00033
00034 #  define DEC  10
00035 #  define HEX  16
00036 #  define OCT   8
00037 #  define BIN   2
00038
00039 /* F("...") on Arduino returns __FlashStringHelper* to keep string literals in
00040  * flash.  On non-Arduino it is the identity macro, resolving to const char*. */
00041 class __FlashStringHelper {};
00042 #  define F(string_literal) (string_literal)
00043
00044 #endif /* ARDUINO / !ARDUINO */
00045
00046 #endif /* PLATFORM_COMPAT_H */
```

6.61 examples/BasicUsage/README.md File Reference

6.62 examples/Connect/README.md File Reference

6.63 examples/README.md File Reference

6.64 examples/Sign/README.md File Reference

6.65 examples/UsdcSigning/README.md File Reference

6.66 examples/VerifyPin/README.md File Reference

6.67 README.md File Reference

6.68 src/cryptnox-sdk-cpp/fuzz/corpus/README.md File Reference

6.69 src/cryptnox-sdk-cpp/README.md File Reference

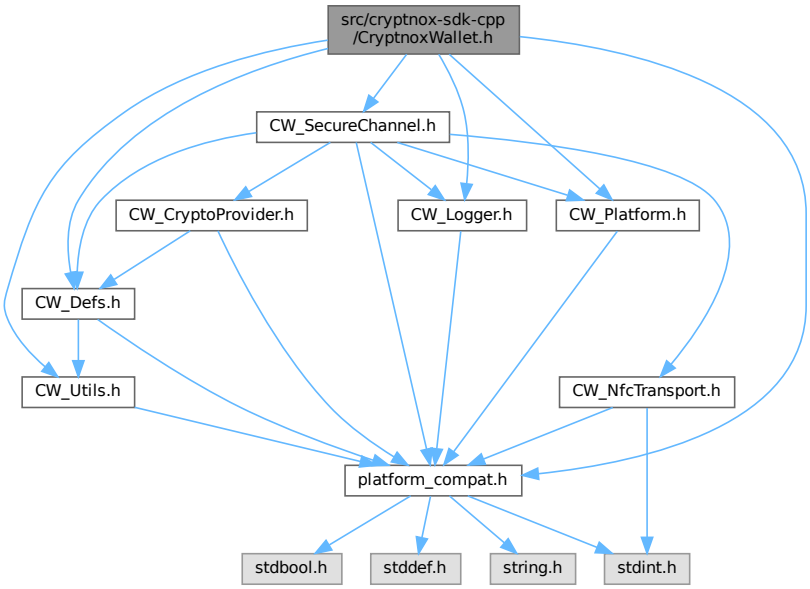
6.70 src/cryptnox-sdk-cpp/CryptnoxWallet.h File Reference

High-level API for interacting with a Cryptnox Hardware Wallet over NFC.

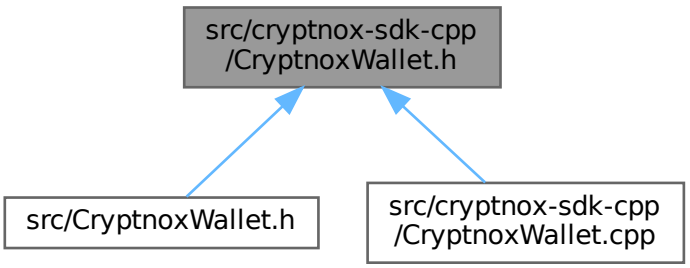
```
#include "platform_compat.h"
#include "CW_Defs.h"
```

```
#include "CW_Logger.h"
#include "CW_Platform.h"
#include "CW_SecureChannel.h"
#include "CW_Utils.h"
```

Include dependency graph for CryptnoxWallet.h:



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



Classes

- struct CW_CardInfo

  Subset of the Cryptnox card info returned by APDU 0x80FA0000.
- struct CW_SignRequest

  Request parameters for CryptnoxWallet::sign.
- struct CW_SignResult

  Result of CryptnoxWallet::sign.

- class CryptnoxWallet

  High-level interface for interacting with a Cryptnox Hardware Wallet over NFC.

Macros

- #define CW_CARD_NAME_MAX_LEN (20U)

  Max name length stored on a Cryptnox card (per card spec).
- #define CW_CARD_EMAIL_MAX_LEN (60U)

  Max email length stored on a Cryptnox card (per card spec).

6.70.1 Detailed Description

High-level API for interacting with a Cryptnox Hardware Wallet over NFC.
Declares CryptnoxWallet, the main entry point for application code. The class wires together the four abstract adapters supplied by the host integration (NFC transport, crypto provider, logger, platform) and exposes the wallet operations: card connection, secure channel establishment, card info retrieval, PIN verification, transaction signing, and user-data writing.

See also

CW_NfcTransport

CW_CryptoProvider

CW_Logger

CW_Platform

Definition in file CryptnoxWallet.h.

6.70.2 Macro Definition Documentation

6.70.2.1 CW_CARD_EMAIL_MAX_LEN

```
#define CW_CARD_EMAIL_MAX_LEN (60U)
```

Max email length stored on a Cryptnox card (per card spec).
Definition at line 45 of file CryptnoxWallet.h.
Referenced by CryptnoxWallet::getCardInfo().

6.70.2.2 CW_CARD_NAME_MAX_LEN

```
#define CW_CARD_NAME_MAX_LEN (20U)
```

Max name length stored on a Cryptnox card (per card spec).
Definition at line 42 of file CryptnoxWallet.h.
Referenced by CryptnoxWallet::getCardInfo().

6.71 CryptnoxWallet.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
00022
00023 #ifndef CRYPTNOXWALLET_H
00024 #define CRYPTNOXWALLET_H
00025
00026 /**********************************************************************
00027  * 1. Included files
00028  **********************************************************************/
00029
00030 #include "platform_compat.h"
00031 #include "CW_Defs.h"
```

```
00032 #include "CW_Logger.h"
00033 #include "CW_Platform.h"
00034 #include "CW_SecureChannel.h"
00035 #include "CW_Utils.h"
00036
00037 /******************************************************************
00038  * 2. Typedefs / structs (sign API)
00039  ******************************************************************/
00040
00042 #define CW_CARD_NAME_MAX_LEN  (20U)
00043
00045 #define CW_CARD_EMAIL_MAX_LEN (60U)
00046
00055 struct CW_CardInfo {
00056     char name[CW_CARD_NAME_MAX_LEN + 1U];
00057     char email[CW_CARD_EMAIL_MAX_LEN + 1U];
00058
00059     CW_CardInfo() {
00060         name[0]  = '\0';
00061         email[0] = '\0';
00062     }
00063 };
00064
00074 struct CW_SignRequest {
00075     CW_SecureSession& session;
00076     uint8_t keyType;
00077     uint8_t signatureType;
00078     uint8_t pin[CW_MAX_PIN_LENGTH];
00079     bool pinLessMode;
00080     const uint8_t* hash;
00081     uint8_t hashLength;
00082     const uint8_t* derivePath;
00083     uint8_t derivePathLength;
00084
00092     explicit CW_SignRequest(CW_SecureSession& sess,
00093                             uint8_t kType   = CW_SIGN_CURR_K1,
00094                             uint8_t sigType = CW_SIGN_SIG_ECDSA_LOW_S,
00095                             bool pinless    = CW_SIGN_WITH_PIN)
00096         : session(sess), keyType(kType), signatureType(sigType),
00097           pinLessMode(pinless), hash(NULL), hashLength(0U),
00098           derivePath(NULL), derivePathLength(0U) {
00099         memset(pin, 0U, sizeof(pin));
00100     }
00101
00103     ~CW_SignRequest() {
00104         CW_Utils::secure_wipe(pin, sizeof(pin));
00105     }
00106 };
00107
00116 struct CW_SignResult {
00117     uint8_t signature[CW_RAW_SIGNATURE_SIZE];
00118     uint8_t errorCode;
00119
00121     CW_SignResult() : errorCode(CW_NOK) {
00122         memset(signature, 0U, sizeof(signature));
00123     }
00124 };
00125
00126 /******************************************************************
00127  * 3. CryptnoxWallet class
00128  ******************************************************************/
00129
00160 class CryptnoxWallet {
00161 public:
00170     CryptnoxWallet(CW_NfcTransport& driver, CW_Logger& logger,
00171                    CW_CryptoProvider& crypto, CW_Platform& platform);
00172
00173     CryptnoxWallet(const CryptnoxWallet&) = delete;
00174     CryptnoxWallet& operator=(const CryptnoxWallet&) = delete;
00175
00180     bool begin();
00181
00200     bool connect(CW_SecureSession& session);
00201
00217     bool establishSecureChannel(CW_SecureSession& session);
00218
00232     void disconnect(CW_SecureSession& session);
00233
00245     bool getCardInfo(CW_SecureSession& session, CW_CardInfo* info = NULL);
00246
00267     bool verifyPin(CW_SecureSession& session, const uint8_t* pin, uint8_t pinLength);
00268
00298     CW_SignResult sign(CW_SignRequest& request);
00299
00309     bool writeUserData(CW_SecureSession& session, uint8_t slot,
00310                        const uint8_t* data, uint16_t dataLength);
00311
```

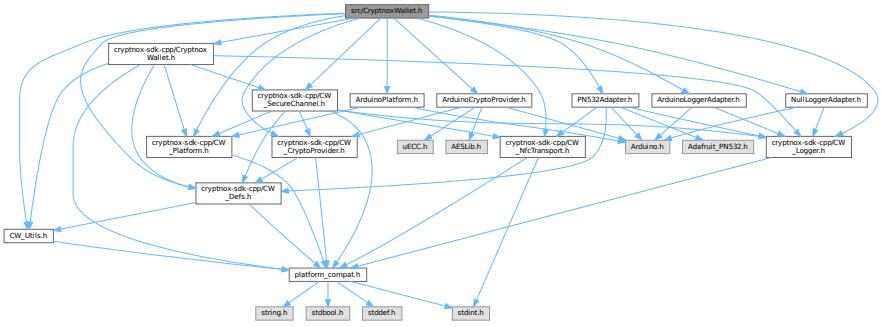
```
00323     static bool parseDerSignature(const uint8_t* der, uint8_t derLength,
00324                                   uint8_t* r, uint8_t& rLength,
00325                                   uint8_t* s, uint8_t& sLength);
00326
00327 private:
00328     CW_Logger&       _logger;
00329     CW_Platform&     _platform;
00330     CW_SecureChannel _secure;
00331
00332     bool isSecureChannelOpen(const CW_SecureSession& session) const;
00333     bool printPN532FirmwareVersion();
00334
00335     /* Sign helper methods */
00336     bool validateSignRequest(const CW_SignRequest& request, CW_SignResult& result);
00337     void buildSignPayload(const CW_SignRequest& request, uint8_t* data, uint16_t& dataLength);
00338     bool sendSignApdu(CW_SignRequest& request, const uint8_t* data, uint16_t dataLength,
00339                       uint8_t* derResponse, uint16_t& derLength, CW_SignResult& result);
00340     bool extractRawSignature(const uint8_t* derResponse, uint16_t derLength, CW_SignResult& result);
00341     void debugPrintSignature(const uint8_t* signature);
00342 };
00343
00344 #endif // CRYPTNOXWALLET_H
```

6.72 src/CryptnoxWallet.h File Reference

Umbrella include and module-group anchor for the CryptnoxWallet Arduino library.

```
#include "cryptnox-sdk-cpp/CW_Defs.h"
#include "cryptnox-sdk-cpp/CW_NfcTransport.h"
#include "cryptnox-sdk-cpp/CW_Logger.h"
#include "cryptnox-sdk-cpp/CW_CryptoProvider.h"
#include "cryptnox-sdk-cpp/CW_Platform.h"
#include "cryptnox-sdk-cpp/CW_SecureChannel.h"
#include "cryptnox-sdk-cpp/CW_Utils.h"
#include "cryptnox-sdk-cpp/CryptnoxWallet.h"
#include "ArduinoLoggerAdapter.h"
#include "NullLoggerAdapter.h"
#include "ArduinoCryptoProvider.h"
#include "ArduinoPlatform.h"
#include "PN532Adapter.h"
```

Include dependency graph for CryptnoxWallet.h:



6.72.1 Detailed Description

Umbrella include and module-group anchor for the CryptnoxWallet Arduino library.
Including this single header pulls in the platform-independent core SDK (the `cryptnox-sdk-cpp` submodule) together with all Arduino-specific concrete adapters (PN532 transport, AESLib/↩ SHA512/micro-ecc crypto provider, Serial logger, null logger). Application sketches should not include the individual headers directly — this umbrella keeps the include surface stable across Arduino IDE versions and library-manager installs.

Header pulled in	Role
cryptnox-sdk-cpp/CW_Defs.h	Shared constants, CW_SecureSession struct, group definitions
cryptnox-sdk-cpp/CW_NfcTransport.h	Abstract NFC transport interface
cryptnox-sdk-cpp/CW_Logger.h	Abstract logging interface
cryptnox-sdk-cpp/CW_CryptoProvider.h	Abstract crypto interface (SHA, AES-CBC, ECDH, RNG, ECDSA verify)
cryptnox-sdk-cpp/CW_Platform.h	Abstract platform interface (sleep_ms)
cryptnox-sdk-cpp/CW_SecureChannel.h	Secure channel protocol implementation
cryptnox-sdk-cpp/CW_Utils.h	Secure compare, secure wipe, bounded mem-cpy
cryptnox-sdk-cpp/CryptnoxWallet.h	High-level card API
ArduinoLoggerAdapter.h	Concrete CW_Logger over HardwareSerial (dev/debug)
NullLoggerAdapter.h	Silent no-op CW_Logger (production — see LOW-03)
ArduinoCryptoProvider.h	Concrete CW_CryptoProvider (AESLib / SHA-512 / micro-ecc / RA4M1 TRNG)
ArduinoPlatform.h	Concrete CW_Platform over Arduino's blocking delay()
PN532Adapter.h	Concrete CW_NfcTransport over Adafruit_↩ PN532

Minimal sketch

```
#include <CryptnoxWallet.h>

NullLoggerAdapter     logger;             // silent in production
ArduinoCryptoProvider crypto;             // AES + SHA + ECDH + TRNG
PN532Adapter          nfc(logger, 10);    // SPI on SS=10
ArduinoPlatform       platform;
CryptnoxWallet        wallet(nfc, logger, crypto, platform);

void setup() {
    wallet.begin();
}
```

See also

CryptnoxWallet
CW_NfcTransport
CW_CryptoProvider
CW_Logger

Definition in file CryptnoxWallet.h.

6.73 CryptnoxWallet.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
00053
00070 #pragma once
00071
```

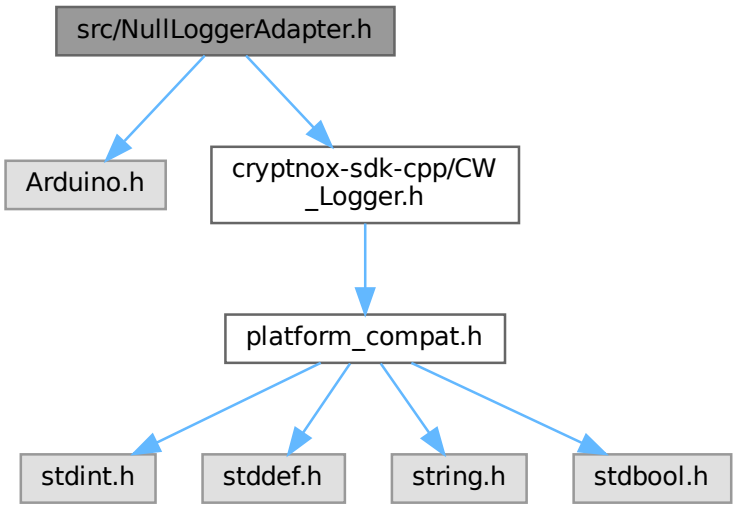
```
00072 #include "cryptnox-sdk-cpp/CW_Defs.h"
00073 #include "cryptnox-sdk-cpp/CW_NfcTransport.h"
00074 #include "cryptnox-sdk-cpp/CW_Logger.h"
00075 #include "cryptnox-sdk-cpp/CW_CryptoProvider.h"
00076 #include "cryptnox-sdk-cpp/CW_Platform.h"
00077 #include "cryptnox-sdk-cpp/CW_SecureChannel.h"
00078 #include "cryptnox-sdk-cpp/CW_Utils.h"
00079 #include "cryptnox-sdk-cpp/CryptnoxWallet.h"
00080 #include "ArduinoLoggerAdapter.h"
00081 #include "NullLoggerAdapter.h"
00082 #include "ArduinoCryptoProvider.h"
00083 #include "ArduinoPlatform.h"
00084 #include "PN532Adapter.h"
```

6.74 src/NullLoggerAdapter.h File Reference

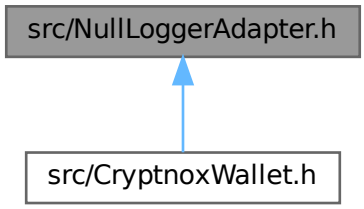
Silent `CW_Logger` for production firmware.

```
#include <Arduino.h>
#include "cryptnox-sdk-cpp/CW_Logger.h"
```

Include dependency graph for NullLoggerAdapter.h:



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



Classes

- class NullLoggerAdapter

  No-op `CW_Logger` — guarantees nothing reaches the serial port.

6.74.1 Detailed Description

Silent `CW_Logger` for production firmware.
Declares NullLoggerAdapter, a no-op logger that drops every call. Use this in shipping firmware instead of ArduinoLoggerAdapter so the SDK's debug traces never reach the UART.

See also

CW_Logger

ArduinoLoggerAdapter — development-time counterpart.

Definition in file NullLoggerAdapter.h.

6.75 NullLoggerAdapter.h

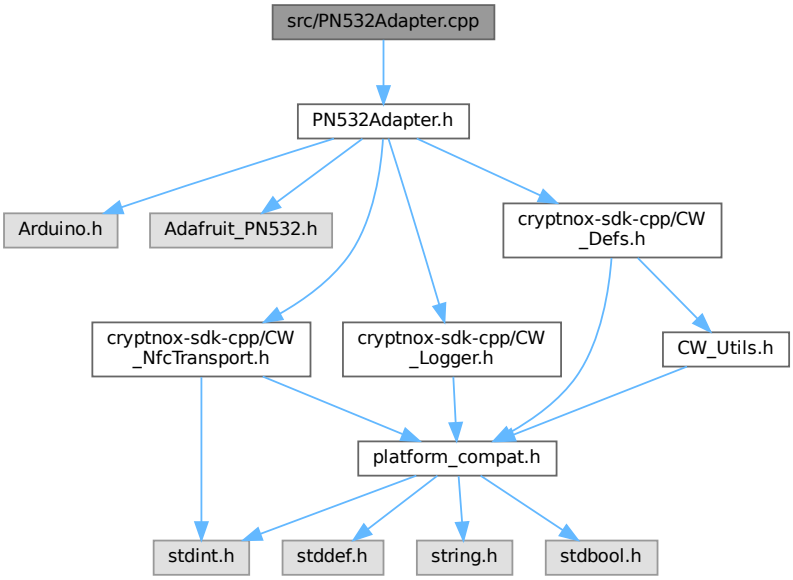
Go to the documentation of this file.

```
00001 /*
00002  * SPDX-License-Identifier: LGPL-3.0-or-later
00003  * Copyright (c) 2026 Cryptnox SA
00004  */
00005
00017
00018 #ifndef NULLLOGGERADAPTER_H
00019 #define NULLLOGGERADAPTER_H
00020
00021 #include <Arduino.h>
00022 #include "cryptnox-sdk-cpp/CW_Logger.h"
00023
00050 class NullLoggerAdapter : public CW_Logger {
00051 public:
00052     NullLoggerAdapter() = default;
00053     ~NullLoggerAdapter() override = default;
00054
00055     NullLoggerAdapter(const NullLoggerAdapter&) = delete;
00056     NullLoggerAdapter& operator=(const NullLoggerAdapter&) = delete;
00057
00059     bool begin(unsigned long /*baudRate*/ = 115200UL) override { return true; }
00060
00061     void print(const __FlashStringHelper* /*str*/) override {}
00062     void print(const char* /*str*/) override {}
00063     void print(char /*c*/) override {}
00064     void print(uint8_t /*value*/, int /*base*/ = DEC) override {}
00065     void print(uint16_t /*value*/, int /*base*/ = DEC) override {}
00066     void print(uint32_t /*value*/, int /*base*/ = DEC) override {}
00067     void print(int /*value*/, int /*base*/ = DEC) override {}
00068
00069     void println() override {}
00070     void println(const __FlashStringHelper* /*str*/) override {}
00071     void println(const char* /*str*/) override {}
00072     void println(char /*c*/) override {}
00073     void println(uint8_t /*value*/, int /*base*/ = DEC) override {}
00074     void println(uint16_t /*value*/, int /*base*/ = DEC) override {}
00075     void println(uint32_t /*value*/, int /*base*/ = DEC) override {}
00076     void println(int /*value*/, int /*base*/ = DEC) override {}
00077 };
00078
00079 #endif // NULLLOGGERADAPTER_H
```

6.76 src/PN532Adapter.cpp File Reference

Implementation of the PN532 transport adapter.
`#include "PN532Adapter.h"`

Include dependency graph for PN532Adapter.cpp:



6.76.1 Detailed Description

Implementation of the PN532 transport adapter.
Forwards each CW_NfcTransport call to the underlying `Adafruit_PN532` instance, plus:

- upfront APDU-length bound check (HIGH-05),
- optional response hex-dump when `CW_DEBUG_LOGGING` is set,
- firmware-version pretty-printer used by CryptnoxWallet::begin().

Doxygen documentation lives on the declarations in PN532Adapter.h.
Definition in file PN532Adapter.cpp.

6.77 PN532Adapter.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
00018
00019 #include "PN532Adapter.h"
00020
00021 // cppcheck-suppress misra-c2012-12.3 -- C++: member initializer-list commas are not the comma
      operator
00022 PN532Adapter::PN532Adapter(CW_Logger& logger, uint8_t ssPin, SPIClass *theSPI)
00023     : _logger(&logger), _interface(PN532Interface::SPI_HARDWARE),
00024       _nfc(new Adafruit_PN532(ssPin, theSPI)) {}
00025
00026 // cppcheck-suppress misra-c2012-12.3 -- C++: member initializer-list commas are not the comma
      operator
00027 PN532Adapter::PN532Adapter(CW_Logger& logger, uint8_t clk, uint8_t miso, uint8_t mosi, uint8_t ss)
00028     : _logger(&logger), _interface(PN532Interface::SPI_SOFTWARE),
00029       _nfc(new Adafruit_PN532(clk, miso, mosi, ss)) {}
00030
00031 // cppcheck-suppress misra-c2012-12.3 -- C++: member initializer-list commas are not the comma
      operator
```

```
00032 PN532Adapter::PN532Adapter(CW_Logger& logger, uint8_t irqPin, uint8_t resetPin, TwoWire *wire)
00033     : _logger(&logger), _interface(PN532Interface::I2C),
00034       _nfc(new Adafruit_PN532(irqPin, resetPin, wire)) {}
00035
00036 // cppcheck-suppress misra-c2012-12.3 -- C++: member initializer-list commas are not the comma
      operator
00037 PN532Adapter::PN532Adapter(CW_Logger& logger, uint8_t resetPin, HardwareSerial *uartSerial)
00038     : _logger(&logger), _interface(PN532Interface::UART),
00039       _nfc(new Adafruit_PN532(resetPin, uartSerial)) {}
00040
00041 PN532Adapter::~PN532Adapter() {
00042     if (_nfc) {
00043         delete _nfc;
00044         _nfc = NULL;
00045     }
00046 }
00047
00048 bool PN532Adapter::begin() {
00049     _nfc->begin();
00050     return _nfc->getFirmwareVersion() != 0;
00051 }
00052
00053 bool PN532Adapter::sendAPDU(const uint8_t* apdu, uint8_t apduLen,
00054                             uint8_t* response, uint8_t& responseLen) {
00055     /* HIGH-05: the uint8_t apduLen already constrains the APDU to the
00056      * PN532's ISO-DEP 255-byte limit at the type level; no runtime check
00057      * needed. */
00058
00059     /* Adafruit_PN532::inDataExchange takes a uint8_t* for the response
00060      * length (in = capacity, out = actual), matching our uint8_t& on
00061      * the CW_NfcTransport contract -- pass it through directly. */
00062     bool success = _nfc->inDataExchange(const_cast<uint8_t*>(apdu), apduLen,
00063                                         response, &responseLen);
00064     if (!success) {
00065         _logger->println(F("APDU exchange failed!"));
00066         return false;
00067     }
00068
00069 #if CW_DEBUG_LOGGING
00070     _logger->print(F("APDU response ("));
00071     _logger->print(responseLen);
00072     _logger->println(F(" bytes):"));
00073
00074     for (uint8_t i = 0; i < responseLen; i++) {
00075         _logger->print(F("0x"));
00076         if (response[i] < 16) { _logger->print(F("0")); }
00077         _logger->print(response[i], HEX);
00078         _logger->print(F(" "));
00079         if (((i + 1) % 16 == 0) && ((i + 1) != responseLen)) { _logger->println(); }
00080     }
00081     _logger->println();
00082 #endif /* CW_DEBUG_LOGGING */
00083
00084     return true;
00085 }
00086
00087 bool PN532Adapter::sendAPDULarge(const uint8_t* apdu, uint8_t apduLen,
00088                                  uint8_t* response, uint16_t& responseLen) {
00089     /* Adafruit_PN532::inDataExchange16 is provided by the
00090      * Adafruit_PN532_extended_frame.patch. It handles both PN532 normal
00091      * frames (1-byte LEN, <=255-byte payload) and PN532 extended frames
00092      * (0xFF 0xFF + 16-bit LEN, up to ~436-byte payload). The Cryptnox
00093      * card's GET_MANUFACTURER_CERTIFICATE per-page answer exceeds the
00094      * normal-frame ceiling on cards whose cert is ~411 bytes, so
00095      * delegating to the default uint8_t sendAPDU would lose the data. */
00096     bool success = _nfc->inDataExchange16(const_cast<uint8_t*>(apdu), apduLen,
00097                                           response, &responseLen);
00098     if (!success) {
00099         _logger->println(F("APDU exchange (large) failed!"));
00100         return false;
00101     }
00102
00103 #if CW_DEBUG_LOGGING
00104     _logger->print(F("APDU response ("));
00105     _logger->print(responseLen);
00106     _logger->println(F(" bytes, large):"));
00107
00108     for (uint16_t i = 0; i < responseLen; i++) {
00109         _logger->print(F("0x"));
00110         if (response[i] < 16) { _logger->print(F("0")); }
00111         _logger->print(response[i], HEX);
00112         _logger->print(F(" "));
00113         if (((i + 1) % 16 == 0) && ((i + 1) != responseLen)) { _logger->println(); }
00114     }
00115     _logger->println();
00116 #endif /* CW_DEBUG_LOGGING */
00117
```

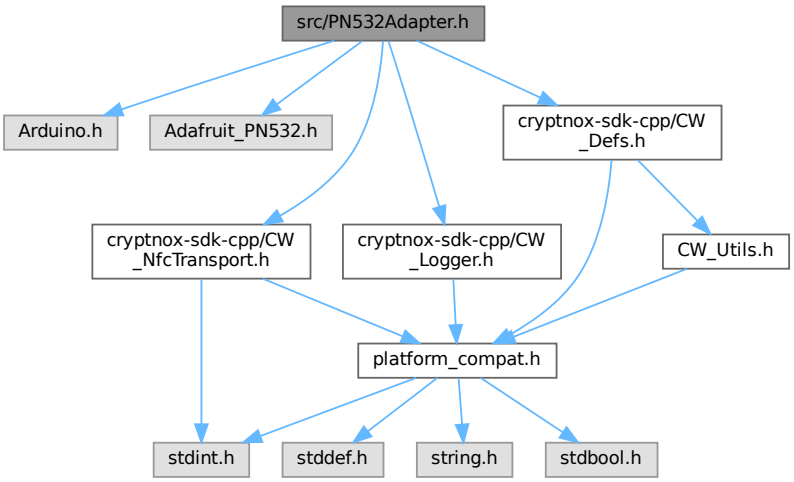
```
00118     return true;
00119 }
00120
00121 bool PN532Adapter::inListPassiveTarget() {
00122     return _nfc->inListPassiveTarget();
00123 }
00124
00125 void PN532Adapter::resetReader() {
00126     _nfc->SAMConfig();
00127 }
00128
00129 bool PN532Adapter::printFirmwareVersion() {
00130     uint32_t versionData = _nfc->getFirmwareVersion();
00131
00132     if (versionData == 0) {
00133         _logger->println(F("PN532 not found!"));
00134         return false;
00135     }
00136
00137     uint8_t ic       = (versionData » 24U) & 0xFFU;
00138     uint8_t verMajor = (versionData » 16U) & 0xFFU;
00139     uint8_t verMinor = (versionData »  8U) & 0xFFU;
00140     uint8_t flags    =  versionData         & 0xFFU;
00141     bool first       = true;
00142
00143     _logger->println(F("PN532 information"));
00144     _logger->print(F(" +- Raw firmware: 0x"));
00145     _logger->println(versionData, HEX);
00146
00147     _logger->print(F(" +- IC Chip: "));
00148     _logger->println((ic == 0x32U) ? F("PN532") : F("Unknown"));
00149
00150     _logger->print(F(" +- Firmware: "));
00151     _logger->print(verMajor);
00152     _logger->print(F("."));
00153     _logger->println(verMinor);
00154
00155     _logger->print(F(" +- Features: "));
00156     if ((flags & 0x01U) != 0U) { _logger->print(F("MIFARE")); first = false; }
00157     if ((flags & 0x02U) != 0U) {
00158         if (!first) { _logger->print(F(" + ")); }
00159         _logger->print(F("ISO-DEP")); first = false;
00160     }
00161     if ((flags & 0x04U) != 0U) {
00162         if (!first) { _logger->print(F(" + ")); }
00163         _logger->print(F("FeliCa")); first = false;
00164     }
00165     if (first) { _logger->print(F("Unknown")); }
00166
00167     _logger->print(F(" (0x"));
00168     _logger->print(flags, HEX);
00169     _logger->println(F(")"));
00170
00171     _nfc->SAMConfig();
00172     return true;
00173 }
```

6.78 src/PN532Adapter.h File Reference

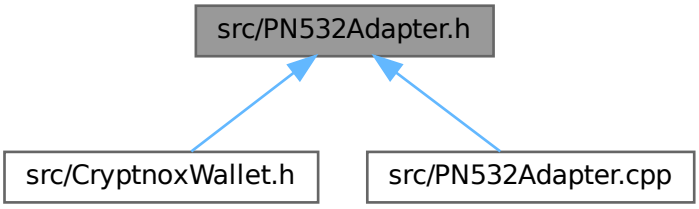
Concrete `CW_NfcTransport` over Adafruit_PN532 (SPI / I2C / UART).

```
#include <Arduino.h>
#include <Adafruit_PN532.h>
#include "cryptnox-sdk-cpp/CW_NfcTransport.h"
#include "cryptnox-sdk-cpp/CW_Logger.h"
#include "cryptnox-sdk-cpp/CW_Defs.h"
```

Include dependency graph for PN532Adapter.h:



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



Classes

- class PN532Adapter
  `CW_NfcTransport` *implementation over the Adafruit_PN532 driver.*

Enumerations

- enum class PN532Interface { PN532Interface::SPI_HARDWARE , PN532Interface::SPI_SOFTWARE , PN532Interface::I2C , PN532Interface::UART }
  Physical wiring used to reach the PN532 reader.

6.78.1 Detailed Description

Concrete `CW_NfcTransport` over Adafruit_PN532 (SPI / I2C / UART).
Declares PN532Adapter, the Arduino-side concrete transport that the host integration injects into CryptnoxWallet. Wraps the four wiring variants the `Adafruit_PN532` library supports — hardware SPI, software (bit-banged) SPI, I2C, and UART — behind the single platform-independent CW_NfcTransport interface so the higher layers never need to know how the reader is actually wired.

See also

CW_NfcTransport

CryptnoxWallet

Definition in file PN532Adapter.h.

6.79 PN532Adapter.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
00021
00022 #ifndef PN532ADAPTER_H
00023 #define PN532ADAPTER_H
00024
00025 #include <Arduino.h>
00026 #include <Adafruit_PN532.h>
00027 #include "cryptnox-sdk-cpp/CW_NfcTransport.h"
00028 #include "cryptnox-sdk-cpp/CW_Logger.h"
00029 #include "cryptnox-sdk-cpp/CW_Defs.h"
00030
00047 enum class PN532Interface {
00048     SPI_HARDWARE,
00049     SPI_SOFTWARE,
00050     I2C,
00051     UART
00052 };
00053
00086 class PN532Adapter : public CW_NfcTransport {
00087 public:
00099     PN532Adapter(CW_Logger& logger, uint8_t ssPin, SPIClass *theSPI = &SPI);
00100
00114     PN532Adapter(CW_Logger& logger, uint8_t clk, uint8_t miso, uint8_t mosi, uint8_t ss);
00115
00127     PN532Adapter(CW_Logger& logger, uint8_t irqPin, uint8_t resetPin, TwoWire *wire = &Wire);
00128
00139     PN532Adapter(CW_Logger& logger, uint8_t resetPin, HardwareSerial *uartSerial);
00140
00142     ~PN532Adapter() override;
00143
00144     PN532Adapter(const PN532Adapter&) = delete;
00145     PN532Adapter& operator=(const PN532Adapter&) = delete;
00146
00149
00159     bool begin() override;
00160
00181     bool sendAPDU(const uint8_t* apdu, uint8_t apduLen,
00182                   uint8_t* response, uint8_t& responseLen) override;
00183
00201     bool sendAPDULarge(const uint8_t* apdu, uint8_t apduLen,
00202                        uint8_t* response, uint16_t& responseLen) override;
00203
00212     bool inListPassiveTarget() override;
00213
00221     void resetReader() override;
00222
00232     bool printFirmwareVersion() override;
00234
00235 private:
00236     CW_Logger*      _logger;
00237     PN532Interface  _interface;
00238     Adafruit_PN532* _nfc;
00239 };
00240
00241 #endif // PN532ADAPTER_H
```

Chapter 7

Examples

7.1 BasicUsage.ino

Example demonstrating the use of CryptnoxWallet with a PN532 module on Arduino.
Example demonstrating the use of CryptnoxWallet with a PN532 module on Arduino.This sketch initializes the PN532 NFC reader using the CryptnoxWallet class. It continuously detects NFC/ISO-DEP cards and processes wallet-specific APDU commands with granular step-by-step control.
Select the communication interface by defining USE_SPI or USE_I2C below.

```
/*
 * SPDX-License-Identifier: LGPL-3.0-or-later
 * Copyright (c) 2026 Cryptnox SA
 */


#include <CryptnoxWallet.h>

/* ========================================================================
 * 1. Interface Selection
 * ------------------------------------------------------------------------
 * Uncomment exactly ONE of the two lines below to choose the PN532 interface.
 * ======================================================================== */
#define USE_SPI
// #define USE_I2C

/* ========================================================================
 * 2. Pin Configuration
 * ======================================================================== */
#if defined(USE_SPI)

#include <SPI.h>

#define PN532_SS    (10U)

/* PRODUCTION NOTE (LOW-03): ArduinoLoggerAdapter streams all log output over
 * USB-CDC Serial, which is visible to any attached host process.  For firmware
 * that is shipped to end-users, replace the two lines below with:
 *   NullLoggerAdapter serialAdapter;
 * and remove the serialAdapter.begin() call in setup().  NullLoggerAdapter is
 * a no-op and incurs zero code-size or runtime overhead. */
ArduinoLoggerAdapter serialAdapter;
PN532Adapter nfc(serialAdapter, PN532_SS, &SPI);

#elif defined(USE_I2C)

#include <Wire.h>

#define PN532_IRQ   (2U)

#define PN532_RST   (3U)

/* PRODUCTION NOTE (LOW-03): see comment in the USE_SPI block above. */
ArduinoLoggerAdapter serialAdapter;
PN532Adapter nfc(serialAdapter, PN532_IRQ, PN532_RST, &Wire);

#else
#error "Please define USE_SPI or USE_I2C to select the PN532 interface."
#endif

#define DEFAULT_PIN       "000000000"
#define DEFAULT_PIN_LEN   (sizeof(DEFAULT_PIN) - 1U)

ArduinoCryptoProvider cryptoProvider;
```

```
ArduinoPlatform       platform;
CryptnoxWallet wallet(nfc, serialAdapter, cryptoProvider, platform);

void setup() {
    serialAdapter.begin(115200);

    /* Arduino R4: Wait 1s to get Serial ready */
    delay(1000);

#if defined(USE_SPI)
    /* Initialize SPI bus */
    SPI.begin();
#elif defined(USE_I2C)
    /* Initialize I2C bus */
    Wire.begin();
#endif

    /* Initialize the PN532 module -- halt on failure to avoid silent errors */
    if (!wallet.begin()) {
        serialAdapter.println(F("PN532 init failed"));
        while(1);
    }
}

void loop() {

    /* Step 1: Connect to card and establish secure channel */
    CW_SecureSession session;
    if (wallet.connect(session)) {
        serialAdapter.println(F("Card connected and secure channel established"));

        /* Step 2: Sign a test hash (32 bytes of 0x01 for demo purposes) */
        /* NOTE: Card must have a seed loaded (via Python SDK: card.generate_seed(pin) */
        /*       or card.load_seed(seed, pin)) before signing will work.              */
        serialAdapter.println(F("Signing test hash..."));
        uint8_t testHash[CW_HASH_SIZE];
        memset(testHash, 0x01, sizeof(testHash));

        /* Build sign request per CW_SignRequest API.
         * PIN is included in the sign data payload for authentication.
         * Alternatively, call verifyPin() first and omit the PIN here. */
        CW_SignRequest signRequest(session, CW_SIGN_CURR_K1, CW_SIGN_SIG_ECDSA_LOW_S, CW_SIGN_WITH_PIN);
        signRequest.hash = testHash;
        signRequest.hashLength = sizeof(testHash);
        /* Set PIN (must match the PIN used during card.init()) */
        CW_Utils::safe_memcpy(signRequest.pin, sizeof(signRequest.pin),
                              reinterpret_cast<const uint8_t*>(DEFAULT_PIN), DEFAULT_PIN_LEN);

        CW_SignResult signResult = wallet.sign(signRequest);

        if (signResult.errorCode == CW_OK) {
            serialAdapter.println(F("Signature received (64 bytes raw r||s)"));

            /* Print first 8 bytes of R and S for quick visual check */
            serialAdapter.print(F("  R[0..7]: "));
            for (uint8_t i = CW_SIG_R_OFFSET; i < CW_SIG_R_OFFSET + 8U; i++) {
                if (signResult.signature[i] < 0x10U) serialAdapter.print(F("0"));
                serialAdapter.print(signResult.signature[i], HEX);
                serialAdapter.print(F(" "));
            }
            serialAdapter.println();
            serialAdapter.print(F("  S[0..7]: "));
            for (uint8_t i = CW_SIG_S_OFFSET; i < CW_SIG_S_OFFSET + 8U; i++) {
                if (signResult.signature[i] < 0x10U) serialAdapter.print(F("0"));
                serialAdapter.print(signResult.signature[i], HEX);
                serialAdapter.print(F(" "));
            }
            serialAdapter.println();
            serialAdapter.println(F("Card processed successfully"));
        } else {
            serialAdapter.print(F("Sign failed, errorCode: 0x"));
            serialAdapter.println(signResult.errorCode, HEX);
        }

        /* Securely wipe sensitive buffers */
        CW_Utils::secure_wipe(testHash, sizeof(testHash));
        CW_Utils::secure_wipe(signResult.signature, sizeof(signResult.signature));
    }

    /* Always disconnect to reset reader for next card detection */
    wallet.disconnect(session);

    /* Wait before next iteration */
    delay(1000);
}
```

7.2 Connect.ino

Minimal Cryptnox example: open a secure channel and fetch card info.
Minimal Cryptnox example: open a secure channel and fetch card info.Wiring & prerequisites:

- PN532 NFC reader on SPI, with SS on pin PN532_SS_PIN.
- A Cryptnox card initialised (the secure channel itself does not need a PIN, but the card must be programmed; use `cryptnox init`).

What the sketch does in each loop iteration:

1. Connect to the card and establish the secure channel (CryptnoxWallet::connect).
2. On success, ask the card for its info (CryptnoxWallet::getCardInfo).
3. Disconnect.

This sketch never submits a PIN, so it cannot lock the card. It is the safest starting point to validate that the wiring and the secure channel work end to end before moving to `VerifyPin` or `Sign` examples.

```
/*
 * SPDX-License-Identifier: LGPL-3.0-or-later
 * Copyright (c) 2026 Cryptnox SA
 */

#include <CryptnoxWallet.h>
#include <SPI.h>

#define PN532_SS_PIN  (10U)

ArduinoLoggerAdapter   serialAdapter;

PN532Adapter           nfc(serialAdapter, PN532_SS_PIN, &SPI);

ArduinoCryptoProvider  cryptoProvider;

ArduinoPlatform        platform;

CryptnoxWallet         wallet(nfc, serialAdapter, cryptoProvider, platform);

void setup() {
    serialAdapter.begin(115200);
    delay(1000);                              /* Arduino R4: wait for Serial */
    SPI.begin();

    if (!wallet.begin()) {
        serialAdapter.println(F("PN532 init failed"));
        while (1);
    }

    /* Concrete proof the PN532 is up: prints IC chip, firmware version
     * and supported NFC features (MIFARE / ISO-DEP / FeliCa). */
    nfc.printFirmwareVersion();
}

void loop() {
    CW_SecureSession session;

    if (wallet.connect(session)) {
        serialAdapter.println(F("Card connected, secure channel established"));

        CW_CardInfo info;
        if (wallet.getCardInfo(session, &info)) {
            serialAdapter.print(F("Owner name : "));
            serialAdapter.println(info.name);
            serialAdapter.print(F("Owner email: "));
            serialAdapter.println(info.email);
        } else {
            serialAdapter.println(F("getCardInfo failed (channel error or parse error)"));
        }
    } else {
        serialAdapter.println(F("Card not detected or secure channel failed"));
    }

    wallet.disconnect(session);
    delay(1000);
}
```

7.3 Sign.ino

Minimal Cryptnox example: sign a 32-byte hash on the secp256k1 curve.
Minimal Cryptnox example: sign a 32-byte hash on the secp256k1 curve.Wiring & prerequisites:

- PN532 NFC reader on SPI, with SS on pin PN532_SS_PIN.
- A Cryptnox card initialised with a known PIN AND a loaded seed (use the Cryptnox CLI↩ : `cryptnox init` then `cryptnox seed generate`).
- DEMO_PIN must match the PIN set on the card.

What the sketch does in each loop iteration:

1. Connect to the card and establish the secure channel.
2. Sign a 32-byte 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 CW_SIGN_WITH_PIN).
3. Print the result, securely wipe local secrets, disconnect.

Warning

On CW_SIGN_PIN_INCORRECT (0x82) the sketch enters an infinite halt: every wrong PIN attempt decrements the card's retry counter and reaching 0 permanently blocks the PIN. Recovery then requires the PUK via `cryptnox unblock_pin`. Verify DEMO_PIN before retrying.

Note

The PIN "000000000" used by the project examples is a demo placeholder. In real deployments set a strong PIN, never commit source files containing a real PIN, and keep this value out of any version-controlled config.

The hash filled with 0x01 below 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 <CryptnoxWallet.h>
#include <SPI.h>

#define PN532_SS_PIN  (10U)

#define DEMO_PIN      "000000000"

ArduinoLoggerAdapter   serialAdapter;

PN532Adapter           nfc(serialAdapter, PN532_SS_PIN, &SPI);

ArduinoCryptoProvider  cryptoProvider;

ArduinoPlatform        platform;

CryptnoxWallet         wallet(nfc, serialAdapter, cryptoProvider, platform);

void setup() {
    serialAdapter.begin(115200);
    delay(1000);                              /* Arduino R4: wait for Serial */
    SPI.begin();

    if (!wallet.begin()) {
        serialAdapter.println(F("PN532 init failed"));
        while (1);
    }
}

void loop() {
    CW_SecureSession session;
    if (!wallet.connect(session)) {
        serialAdapter.println(F("Card not detected"));
        wallet.disconnect(session);
        delay(1000);
```

```
        return;
    }

    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);
    CW_Utils::safe_memcpy(req.pin, sizeof(req.pin),
                          reinterpret_cast<const uint8_t*>(DEMO_PIN), strlen(DEMO_PIN));

    CW_SignResult sig = wallet.sign(req);
    if (sig.errorCode == CW_OK) {
        serialAdapter.println(F("Signature OK"));

        serialAdapter.print(F("  r = "));
        for (uint8_t i = 0U; i < 32U; i++) {
            uint8_t b = sig.signature[CW_SIG_R_OFFSET + i];
            if (b < 0x10U) { serialAdapter.print(F("0")); }
            serialAdapter.print(b, HEX);
        }
        serialAdapter.println();

        serialAdapter.print(F("  s = "));
        for (uint8_t i = 0U; i < 32U; i++) {
            uint8_t b = sig.signature[CW_SIG_S_OFFSET + i];
            if (b < 0x10U) { serialAdapter.print(F("0")); }
            serialAdapter.print(b, HEX);
        }
        serialAdapter.println();

        serialAdapter.print(F("  errorCode = 0x"));
        serialAdapter.println(sig.errorCode, HEX);
    } else if (sig.errorCode == CW_SIGN_PIN_INCORRECT) {
        /* CRITICAL: do NOT loop -- each wrong PIN attempt burns a retry. */
        serialAdapter.println(F("Wrong PIN -- halting to protect retry counter"));
        CW_Utils::secure_wipe(hash, sizeof(hash));
        CW_Utils::secure_wipe(sig.signature, sizeof(sig.signature));
        wallet.disconnect(session);
        while (1);
    } else {
        serialAdapter.print(F("Sign failed: 0x"));
        serialAdapter.println(sig.errorCode, HEX);
    }

    CW_Utils::secure_wipe(hash, sizeof(hash));
    CW_Utils::secure_wipe(sig.signature, sizeof(sig.signature));
    wallet.disconnect(session);
    delay(1000);
}
```

7.4 UsdcSigning.ino

Send an ERC20 USDC transaction on Ethereum (EIP-1559) via Arduino, signed on-device using a Cryptnox smart card over NFC (PN532).
Send an ERC20 USDC transaction on Ethereum (EIP-1559) via Arduino, signed on-device using a Cryptnox smart card over NFC (PN532).Demonstrates:

- Connecting to WiFi
- RLP encoding of an unsigned and signed EIP-1559 transaction
- Keccak-256 hashing
- Signing the transaction hash with a Cryptnox card via PN532
- Determining yParity via the Ethereum ecrecover precompile (eth_call)
- Sending the signed transaction via JSON-RPC

```
/*
 * SPDX-License-Identifier: LGPL-3.0-or-later
 * Copyright (c) 2026 Cryptnox SA
```

```
 */


#include <Arduino.h>
#include <SPI.h>
#include "keccak256.h"
#include <WiFiS3.h>
#include <ArduinoHttpClient.h>
#include "util.h"
#include "config.h"
#include <CryptnoxWallet.h>

#define PN532_SS_PIN  (10U)

/* Fallback PIN -- define CARD_PIN and CARD_PIN_LEN in config.h to override.
 * L-04: hardcoded PIN lives in flash, recoverable via SWD/JTAG -- OK for
 * demo, not for prod. See config.template.h for safer patterns. */
#ifndef CARD_PIN
#  define CARD_PIN     "000000000"
#  define CARD_PIN_LEN (9U)
#endif

/* Default RPC path -- PublicNode accepts JSON-RPC at root.
 * Infura requires /v3/{PROJECT_ID}: define RPC_PATH in config.h for that case. */
#ifndef RPC_PATH
#  define RPC_PATH "/"
#endif

/* M-04: TLS server-certificate pinning.
 *
 * Without setCACert(), WiFiSSLClient accepts ANY certificate the RPC endpoint
 * presents -- a network attacker can MITM the connection and feed crafted
 * nonces / gas prices to make the device sign incorrect transactions, or
 * exfiltrate the basic-auth credentials sent to Infura.
 *
 * The default below is ISRG Root X1 (Let's Encrypt), which signs the chains
 * used by PublicNode (the template's default provider). For a different
 * provider (Infura/DigiCert, Cloudflare, etc.), override WIFI_CA_CERT in
 * config.h with the appropriate root in PEM form. */
#ifndef WIFI_CA_CERT
#  define WIFI_CA_CERT \
"-----BEGIN CERTIFICATE-----\n" \
"MIIFazCCA1OgAwIBAgIRAIIQz7DSQONZRGPgu2OCiwAwDQYJKoZIhvcNAQELBQAw\n" \
"TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh\n" \
"cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMTUwNjA0MTEwNDM4\n" \
"WhcNMzUwNjA0MTEwNDM4WjBPMQswCQYDVQQGEwJVUzEpMCcGA1UEChMgSW50ZXJu\n" \
"ZXQgU2VjdXJpdHkgUmVzZWFyY2ggR3JvdXAxFTATBgNVBAMTDElTUkcgUm9vdCBY\n" \
"MTCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAK3oJHP0FDfzm54rVygc\n" \
"h77ct984kIxuPOZXoHj3dcKi/vVqbvYATyjb3miGbESTtrFj/RQSa78f0uoxmyF+\n" \
"0TM8ukj13Xnfs7j/EvEhmkvBioZxaUpmZmyPfjxwv60pIgbz5MDmgK7iS4+3mX6U\n" \
"A5/TR5d8mUgjU+g4rk8Kb4Mu0UlXjIB0ttov0DiNewNwIRt18jA8+o+u3dpjq+sW\n" \
"T8KOEUt+zwvo/7V3LvSye0rgTBIlDHCNAymg4VMk7BPZ7hm/ELNKjD+Jo2FR3qyH\n" \
"B5T0Y3HsLuJvW5iB4YlcNHlsdu87kGJ55tukmi8mxdAQ4Q7e2RCOFvu396j3x+UC\n" \
"B5iPNgiV5+I3lg02dZ77DnKxHZu8A/lJBdiB3QW0KtZB6awBdpUKD9jf1b0SHzUv\n" \
"KBds0pjBqAlkd25HN7rOrFleaJ1/ctaJxQZBKT5ZPt0m9STJEadao0xAH0ahmbWn\n" \
"OlFuhjuefXKnEgV4We0+UXgVCwOPjdAvBbI+e0ocS3MFEvzG6uBQE3xDk3SzynTn\n" \
"jh8BCNAw1FtxNrQHusEwMFxIt4I7mKZ9YIqioymCzLq9gwQbooMDQaHWBfEbwrbw\n" \
"qHyGO0aoSCqI3Haadr8faqU9GY/rOPNk3sgrDQoo//fb4hVC1CLQJ13hef4Y53CI\n" \
"rU7m2Ys6xt0nUW7/vGT1M0NPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV\n" \
"HRMBAf8EBTADAQH/MB0GA1UdDgQWBBR5tFnme7bl5AFzgAiIyBpY9umbbjANBgkq\n" \
"hkiG9w0BAQsFAAOCAgEAVR9YqbyyqFDQDLHYGmkgJykIrGF1XIpu+ILlaS/V9lZL\n" \
"ubhzEFnTIZd+50xx+7LSYK05qAvqFyFWhfFQDlnrzuBZ6brJFe+GnY+EgPbk6ZGQ\n" \
"3BebYhtF8GaV0nxvwuo77x/Py9auJ/GpsMiu/X1+mvoiBOv/2X/qkSsisRcOj/KK\n" \
"NFtY2PwByVS5uCbMiogziUwthDyC3+6WVwW6LLv3xLfHTjuCvjHIInNzktHCgKQ5\n" \
"ORAzI4JMPJ+GslWYHb4phowim57iaztXOoJwTdwJx4nLCgdNbOhdjsnvzqvHu7Ur\n" \
"TkXWStAmzOVyyghqpZXjFaH3pO3JLF+l+/+sKAIuvtd7u+Nxe5AW0wdeRlN8NwdC\n" \
"jNPElpzVmbUq4JUagEiuTDkHzsxHpFKVK7q4+63SM1N95R1NbdWhscdCb+ZAJzVc\n" \
"oyi3B43njTOQ5yOf+1CceWxG1bQVs5ZufpsMljq4Ui0/1ivh+wjChP4kqKOJ2qxq\n" \
"4RgqsahDYVvTH9w7jXbyLeiNdd8XM2w9U/t7y0Ff/9yi0GE44Za4rF2LN9d11TPA\n" \
"mRGunUHBcnWEvgJBQl9nJEiU0Zsnvgc/ubhPgXRR4Xq37Z0j4r7g1SgEEzwxA57d\n" \
"emyPxgcYxn/eR44/KJ4EBs+lVDR3veyJm+kXQ99b21/+jh5Xos1AnX5iItreGCc=\n" \
"-----END CERTIFICATE-----\n"
#endif

/* L-05 -- PRODUCTION: USB-CDC Serial is readable by any host process.
 * Logs leak RPC URL, basic-auth header, nonce, recipient, tx hash, and
 * (if CW_DEBUG_LOGGING=1) secure-channel ciphertext + IVs. Before ship:
 * swap for `NullLoggerAdapter serialAdapter;` and drop Serial.begin(). */
ArduinoLoggerAdapter  serialAdapter;
ArduinoCryptoProvider cryptoProvider;
ArduinoPlatform       platform;
PN532Adapter          nfc(serialAdapter, PN532_SS_PIN, &SPI);
CryptnoxWallet        wallet(nfc, serialAdapter, cryptoProvider, platform);

#define ERC20_TRANSFER_SEL_0  0xa9U
#define ERC20_TRANSFER_SEL_1  0x05U
#define ERC20_TRANSFER_SEL_2  0x9cU
```

```
#define ERC20_TRANSFER_SEL_3  0xbbU

#define ERC20_INDEX_OFFSET 64U

#define YPARITY_UNKNOWN 0xFFU

#define HTTP_OK 200

#define TX_MAX_RETRIES       3U
#define TX_RETRY_DELAY_MS 2000U
#define WIFI_RETRY_MAX      20U
#define HEX_CHAR_BUF_SIZE    3U
#define ECRECOVER_V_PAD_CHARS 62U
#define ECRECOVER_V_BASE    27U

struct Tx2 {
    uint64_t nonce;
    uint64_t maxPriorityFeePerGas;
    uint64_t maxFeePerGas;
    uint64_t gasLimit;
    const char* to;
    uint64_t value;
    const uint8_t* data;
    size_t dataLen;
    uint32_t chainId;
};

static const char hexChars[] = "0123456789abcdef";

static void printHex(const char* label, const uint8_t* data, size_t len) {
    Serial.print(label);
    Serial.print(F(": 0x"));
    char buf[3]; buf[2] = '\0';
    for (size_t i = 0; i < len; i++) {
        buf[0] = hexChars[data[i] >> 4];
        buf[1] = hexChars[data[i] & 0x0f];
        Serial.print(buf);
    }
    Serial.println();
}

#if defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET)
#define AUTH_CRED_BUF_SIZE    128U
#define AUTH_HEADER_BUF_SIZE  200U

/* NEW-3: base64Encode now requires the output capacity and returns false
 * if it cannot write the full (ceil(inputLen/3)*4 + 1) bytes. Prevents
 * silent stack overflow if AUTH_HEADER_BUF_SIZE is later reduced or a
 * caller passes a too-small buffer. */
static bool base64Encode(const char* input, size_t inputLen, char* output, size_t outputCap) {
    static const char kAlphabet[] =
        "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
    const size_t required = (((inputLen + 2U) / 3U) * 4U) + 1U; /* +1 for NUL */
    if (outputCap < required) {
        return false;
    }
    size_t i = 0U;
    size_t o = 0U;
    while (i < inputLen) {
        uint32_t u0 = static_cast<uint32_t>(static_cast<uint8_t>(input[i]));
        uint32_t u1 = 0U;
        uint32_t u2 = 0U;
        if ((i + 1U) < inputLen) {
            u1 = static_cast<uint32_t>(static_cast<uint8_t>(input[i + 1U]));
        }
        if ((i + 2U) < inputLen) {
            u2 = static_cast<uint32_t>(static_cast<uint8_t>(input[i + 2U]));
        }
        output[o] = kAlphabet[static_cast<uint8_t>(u0 » 2U)];
        o++;
        output[o] = kAlphabet[static_cast<uint8_t>(((u0 & 0x03U) « 4U) | (u1 » 4U))];
        o++;
        if ((i + 1U) < inputLen) {
            output[o] = kAlphabet[static_cast<uint8_t>(((u1 & 0x0FU) « 2U) | (u2 » 6U))];
        } else {
            output[o] = '=';
        }
        o++;
        if ((i + 2U) < inputLen) {
            output[o] = kAlphabet[static_cast<uint8_t>(u2 & 0x3FU)];
        } else {
            output[o] = '=';
        }
        o++;
        i += 3U;
    }
    output[o] = '\0';
```

```
    return true;
}

static void buildBasicAuthHeader(char* buf, size_t bufSize) {
    static const char kPrefix[] = "Basic ";
    const size_t prefixLen = sizeof(kPrefix) - 1U;
    char cred[AUTH_CRED_BUF_SIZE];
    // cppcheck-suppress invalidPrintfArgType_s -- macros are char* when defined in config.h; --force
      evaluates this block without knowing their types
    int written = snprintf(cred, sizeof(cred), "%s:%s", RPC_PROJECT_ID, RPC_API_SECRET);
    /* H-04: snprintf returns the number of bytes it WOULD have written
     * (excluding NUL). A value >= sizeof(cred) means the project id +
     * secret were truncated -- sending a truncated Authorization header
     * causes opaque 401s and could leak credentials in retry traces. */
    if ((written < 0) || ((size_t)written >= sizeof(cred))) {
        Serial.println(F("[fatal] RPC_PROJECT_ID + RPC_API_SECRET exceed AUTH_CRED_BUF_SIZE"));
        while (true) { /* halt -- do not send a truncated basic-auth header */ }
    }
    /* NEW-1: ensure the output buffer can hold prefix + base64(cred) + NUL.
     * base64 expands by ~4/3; the helper rejects undersized buffers. */
    if (prefixLen >= bufSize) {
        Serial.println(F("[fatal] AUTH_HEADER_BUF_SIZE too small for prefix"));
        while (true) {}
    }
    (void)CW_Utils::safe_memcpy(reinterpret_cast<uint8_t*>(buf), bufSize,
                                reinterpret_cast<const uint8_t*>(kPrefix), prefixLen);
    if (!base64Encode(cred, strlen(cred), buf + prefixLen, bufSize - prefixLen)) {
        Serial.println(F("[fatal] base64 output exceeds AUTH_HEADER_BUF_SIZE"));
        while (true) {}
    }
}
#endif /* defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET) */

static bool ensureWiFi() {
    if (WiFi.status() == WL_CONNECTED) return true;
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    uint8_t retries = WIFI_RETRY_MAX;
    while ((retries > 0U) && (WiFi.status() != WL_CONNECTED)) {
        retries--;
        delay(500U);
    }
    return WiFi.status() == WL_CONNECTED;
}

/* Each RlpEncodeItem call returns 0 on overflow; bail out immediately so
 * the caller sees rlpLen == 0 and halts before broadcasting garbage. */
#define RLP_ITEM_OR_FAIL(BUF, CAP, OFF, IN, IN_LEN)                          \
    do {                                                                     \
        uint32_t _w = RlpEncodeItem((BUF) + (OFF), (CAP) - (OFF),            \
                                    (IN), (IN_LEN));                         \
        if (_w == 0U) { return 0U; }                                         \
        (OFF) += _w;                                                         \
    } while (0)

static size_t rlpEncodeTxBody(uint8_t* buf, size_t bufCap, const Tx2& tx) {
    size_t off = 0;
    uint8_t tmp[8];
    size_t tmpLen;
    tmpLen = ConvertNumberToUintArray(tmp, tx.chainId);
    RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
    tmpLen = ConvertNumberToUintArray(tmp, tx.nonce);
    RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
    tmpLen = ConvertNumberToUintArray(tmp, tx.maxPriorityFeePerGas);
    RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
    tmpLen = ConvertNumberToUintArray(tmp, tx.maxFeePerGas);
    RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
    tmpLen = ConvertNumberToUintArray(tmp, tx.gasLimit);
    RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
    uint8_t addr[20];
    if (!hexToBytes(tx.to, addr, 20)) {
        Serial.println(F("[fatal] tx.to is not a valid 40-char hex string"));
        while (true) { /* halt to avoid broadcasting a malformed transaction */ }
    }
    RLP_ITEM_OR_FAIL(buf, bufCap, off, addr, 20U);
    tmpLen = ConvertNumberToUintArray(tmp, tx.value);
    RLP_ITEM_OR_FAIL(buf, bufCap, off, tmp, (uint32_t)tmpLen);
    RLP_ITEM_OR_FAIL(buf, bufCap, off, tx.data, (uint32_t)tx.dataLen);
    if (off >= bufCap) { return 0U; }   /* room for the access-list terminator 0xC0 */
    buf[off++] = 0xC0;
    return off;
}

static size_t rlpFinalize(uint8_t* out, size_t outCap, const uint8_t* buf, size_t off) {
    uint8_t header[8];
    size_t header_len = RlpEncodeWholeHeader(header, sizeof(header), off);
    if (header_len == 0U) { return 0U; }
    /* Reject early if the total wouldn't fit (1 type byte + header + body). */
```

```
    if ((1U + header_len + off) > outCap) {
        return 0U;
    }
    size_t out_off = 0U;
    out[out_off++] = 0x02;
    (void)CW_Utils::safe_memcpy(out + out_off, outCap - out_off, header, header_len);
    out_off += header_len;
    (void)CW_Utils::safe_memcpy(out + out_off, outCap - out_off, buf, off);
    return out_off + off;
}

size_t rlpEncodeUnsignedTx(const Tx2& tx, uint8_t* out, size_t outCap) {
    uint8_t buf[1024];
    size_t off = rlpEncodeTxBody(buf, sizeof(buf), tx);
    if (off == 0U) { return 0U; }
    return rlpFinalize(out, outCap, buf, off);
}

size_t rlpEncodeSignedTx(const Tx2& tx, const uint8_t* r, const uint8_t* s, const uint8_t* v,
                         uint8_t* out, size_t outCap) {
    uint8_t buf[1024];
    size_t off = rlpEncodeTxBody(buf, sizeof(buf), tx);
    if (off == 0U) { return 0U; }
    uint32_t w;
    w = RlpEncodeItem(buf + off, sizeof(buf) - off, v, 1U);
    if (w == 0U) { return 0U; }
    off += w;
    uint8_t tmp_r[32];
    size_t tmp_len = trimLeadingZeros(tmp_r, sizeof(tmp_r), r, 32U);
    if (tmp_len == 0U) { return 0U; }
    w = RlpEncodeItem(buf + off, sizeof(buf) - off, tmp_r, (uint32_t)tmp_len);
    if (w == 0U) { return 0U; }
    off += w;
    uint8_t tmp_s[32];
    tmp_len = trimLeadingZeros(tmp_s, sizeof(tmp_s), s, 32U);
    if (tmp_len == 0U) { return 0U; }
    w = RlpEncodeItem(buf + off, sizeof(buf) - off, tmp_s, (uint32_t)tmp_len);
    if (w == 0U) { return 0U; }
    off += w;
    size_t ret = rlpFinalize(out, outCap, buf, off);
    /* L-01: wipe the signature scratch buffers for hygiene uniformity with
     * the rest of the codebase. Signatures are public (broadcast on-chain)
     * so this is style/consistency only, not a real secret leak. */
    CW_Utils::secure_wipe(tmp_r, sizeof(tmp_r));
    CW_Utils::secure_wipe(tmp_s, sizeof(tmp_s));
    return ret;
}

size_t encodeERC20Transfer(uint8_t* out) {
    out[0] = ERC20_TRANSFER_SEL_0; /* transfer(address,uint256) selector byte 0 */
    out[1] = ERC20_TRANSFER_SEL_1; /* transfer(address,uint256) selector byte 1 */
    out[2] = ERC20_TRANSFER_SEL_2; /* transfer(address,uint256) selector byte 2 */
    out[3] = ERC20_TRANSFER_SEL_3; /* transfer(address,uint256) selector byte 3 */
    CW_Utils::secure_wipe(out+4,  12U); /* bytes  4-15: ABI word padding before address (12 zero bytes) */
    if (!hexToBytes(ADDR_TO, out+16, 20)) {  /* bytes 16-35: recipient address (20 bytes)           */
        Serial.println(F("[fatal] ADDR_TO is not a valid 40-char hex string"));
        while (true) { /* halt to avoid broadcasting a wrong-recipient transfer */ }
    }
    CW_Utils::secure_wipe(out+36, 28U); /* bytes 36-63: ABI word padding before amount  (28 zero bytes) */
    out[ERC20_INDEX_OFFSET]   = (uint8_t)((AMOUNT_USDC >> 24U) & 0xFFU);
    out[ERC20_INDEX_OFFSET+1] = (uint8_t)((AMOUNT_USDC >> 16U) & 0xFFU);
    out[ERC20_INDEX_OFFSET+2] = (uint8_t)((AMOUNT_USDC >> 8U)  & 0xFFU);
    out[ERC20_INDEX_OFFSET+3] = (uint8_t)( AMOUNT_USDC         & 0xFFU);
    return 68;
}

bool sendRawTx(const uint8_t* raw, size_t len) {
    static const char requestPrefix[] =
        "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"eth_sendRawTransaction\","
        "\"params\":[\"0x";
    /* Fixed closing of the JSON-RPC body that follows the hex transaction bytes. */
    static const char requestSuffix[] = "\"]}";
    bool sent = false;
    for (uint8_t attempt = 0U; (attempt < TX_MAX_RETRIES) && !sent; attempt++) {
        if (attempt != 0U) {
            delay(TX_RETRY_DELAY_MS);
        }
        if (!ensureWiFi()) {
            Serial.println(F("sendRawTx: WiFi reconnect failed"));
            continue;
        }
        WiFiSSLClient wifiClient;
#ifndef WIFI_DISABLE_CA_PINNING
        /* M-04: pin the RPC server's CA so a network MITM cannot serve a
         * forged certificate and feed crafted nonces/gas prices.
         * Define WIFI_DISABLE_CA_PINNING in config.h to skip this -- DEV ONLY,
         * leaves the connection vulnerable to MITM. */
```

```
        wifiClient.setCACert(WIFI_CA_CERT);
#endif
        HttpClient client(wifiClient, RPC_HOST, RPC_PORT);
        client.beginRequest();
        int err = client.post(RPC_PATH);
        if (err != HTTP_SUCCESS) {
            Serial.print(F("sendRawTx: POST failed, err="));
            Serial.println(err);
            client.stop();
            continue;
        }
        /* Content-Length = prefix + 2 hex chars per raw byte + suffix. */
        client.sendHeader("Content-Type", "application/json");
        client.sendHeader("Content-Length",
            (int)(sizeof(requestPrefix)-1) + 2*(int)len + (int)(sizeof(requestSuffix)-1));
#if defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET)
        {
            char authBuf[AUTH_HEADER_BUF_SIZE];
            buildBasicAuthHeader(authBuf, sizeof(authBuf));
            client.sendHeader("Authorization", authBuf);
        }
#endif
        client.beginBody();
        client.print(requestPrefix);
        /* Encode each raw transaction byte as two hex characters and stream it
         * directly to the HTTP client, avoiding a large intermediate buffer. */
        char byteHexStr[HEX_CHAR_BUF_SIZE];
        byteHexStr[2] = '\0';
        for (size_t i = 0; i < len; i++) {
            byteHexStr[0] = hexChars[raw[i] >> 4];   /* high nibble */
            byteHexStr[1] = hexChars[raw[i] & 0x0f]; /* low nibble  */
            client.print(byteHexStr);
        }
        client.print(requestSuffix);
        client.endRequest();
        int status = client.responseStatusCode();
        String responseBody = client.responseBody();
        Serial.print(F("[RPC] HTTP ")); Serial.println(status);
        /* NEW-2: dumping the full RPC response leaks tx metadata (nonce,
         * recipient, gas) over USB-CDC. Gate behind CW_DEBUG_LOGGING so a
         * production build (CW_DEBUG_LOGGING=0) stays silent. */
#if CW_DEBUG_LOGGING
        Serial.print(F("[RPC] ")); Serial.println(responseBody);
#endif
        bool statusOk = (status == HTTP_OK);
        /* L-03 (accepted): coarse substring search instead of a JSON parser.
         * Not a security issue -- at worst a false positive triggers a retry.
         * ArduinoJson would cost +15-25 KB flash for marginal robustness. */
        bool noJsonError = (responseBody.indexOf("\"error\"") == -1);
        sent = statusOk && noJsonError;
        /* eth_sendRawTransaction returns {"jsonrpc":"2.0","id":1,"result":"0x<txhash>"}.
         * The tx hash is on-chain public info -- extract and print so the user can
         * track the broadcast on a block explorer. */
        if (sent) {
            int r = responseBody.indexOf("\"result\":\"0x");
            if (r >= 0) {
                int start = r + 10; /* skip past "\"result\":\"" */
                int end = responseBody.indexOf("\"", start);
                if (end > start) {
                    Serial.print(F("[tx] hash="));
                    Serial.println(responseBody.substring(start, end));
                }
            }
        }
        client.stop();
    }
    return sent;
}

uint8_t determineYParity(const uint8_t* hash, const uint8_t* r, const uint8_t* s) {
    /* ecrecover calldata: "0x" + hash(64) + v(64) + r(64) + s(64) = 258 chars + NUL */
    char hexBuf[260];
    uint16_t pos = 0U;
    hexBuf[pos++] = '0';
    hexBuf[pos++] = 'x';
    for (uint8_t i = 0U; i < 32U; i++) {
        hexBuf[pos++] = hexChars[hash[i] >> 4];
        hexBuf[pos++] = hexChars[hash[i] & 0x0f];
    }
    /* v field: ECRECOVER_V_PAD_CHARS zero chars, then 1 value byte -- filled per iteration */
    const uint16_t vOffset = pos;
    for (uint8_t i = 0U; i < ECRECOVER_V_PAD_CHARS; i++) {
        hexBuf[pos++] = '0';
    }
    pos += 2U; /* placeholder for v byte */
    for (uint8_t i = 0U; i < 32U; i++) {
        hexBuf[pos++] = hexChars[r[i] >> 4];
```

```
        hexBuf[pos++] = hexChars[r[i] & 0x0f];
    }
    for (uint8_t i = 0U; i < 32U; i++) {
        hexBuf[pos++] = hexChars[s[i] >> 4];
        hexBuf[pos++] = hexChars[s[i] & 0x0f];
    }
    hexBuf[pos] = '\0'; /* pos == 258 */

    static const char requestPrefix[] =
        "{\"jsonrpc\":\"2.0\",\"id\":1,\"method\":\"eth_call\","
        "\"params\":[{\"to\":\"0x0000000000000000000000000000000000000001\","
        "\"data\":\"";
    static const char requestSuffix[] = "\"},\"latest\"]}";
    const int bodyLen = (int)(sizeof(requestPrefix) - 1) + 258 + (int)(sizeof(requestSuffix) - 1);

    uint8_t result = YPARITY_UNKNOWN;
    for (uint8_t yp = 0U; (yp <= 1U) && (result == YPARITY_UNKNOWN); yp++) {
        /* Patch v byte into last two chars of the v field.
         * Ethereum ecrecover: v=27 means yParity=0, v=28 means yParity=1. */
        const uint8_t v = ECRECOVER_V_BASE + yp;
        hexBuf[vOffset + ECRECOVER_V_PAD_CHARS]      = hexChars[(v & 0xFFU) >> 4U];
        hexBuf[vOffset + ECRECOVER_V_PAD_CHARS + 1U] = hexChars[(v & 0xFFU) & 0x0FU];

        if (!ensureWiFi()) {
            Serial.println(F("determineYParity: WiFi reconnect failed"));
            continue;
        }
        WiFiSSLClient wifiClient;
#ifndef WIFI_DISABLE_CA_PINNING
        /* M-04: pin the RPC server's CA so a network MITM cannot serve a
         * forged certificate and feed crafted nonces/gas prices.
         * Define WIFI_DISABLE_CA_PINNING in config.h to skip this -- DEV ONLY,
         * leaves the connection vulnerable to MITM. */
        wifiClient.setCACert(WIFI_CA_CERT);
#endif
        HttpClient client(wifiClient, RPC_HOST, RPC_PORT);
        client.beginRequest();
        int err = client.post(RPC_PATH);
        if (err != HTTP_SUCCESS) {
            Serial.print(F("determineYParity: POST failed, err="));
            Serial.println(err);
            client.stop();
            continue;
        }
        client.sendHeader("Content-Type", "application/json");
        client.sendHeader("Content-Length", bodyLen);
#if defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET)
        {
            char authBuf[AUTH_HEADER_BUF_SIZE];
            buildBasicAuthHeader(authBuf, sizeof(authBuf));
            client.sendHeader("Authorization", authBuf);
        }
#endif
        client.beginBody();
        client.print(requestPrefix);
        client.print(hexBuf);
        client.print(requestSuffix);
        client.endRequest();

        int status = client.responseStatusCode();
        String response = client.responseBody(); /* consume response body */
        client.stop();
        if (status != HTTP_OK) {
            continue;
        }
        int resultIdx = response.indexOf("\"result\"");
        if (resultIdx < 0) {
            continue;
        }
        int hexIdx = response.indexOf("0x", resultIdx);
        if (hexIdx < 0) {
            continue;
        }
        /* H-05: validate the response is long enough before substring(). ecrecover
         * returns a 32-byte (64-hex-char) word; with the "0x" prefix the slice
         * spans hexIdx+2 .. hexIdx+66. A truncated RPC response would silently
         * give us an empty/garbage recovered address and let the loop progress. */
        if (response.length() < (unsigned int)(hexIdx + 66)) {
            continue;
        }
        /* ecrecover returns 32-byte word; address = last 20 bytes = last 40 hex chars */
        String recovered = response.substring(hexIdx + 26, hexIdx + 66);
        Serial.print(F("[ecrecover] v=")); Serial.print(v);
        Serial.print(F(" recovered=0x")); Serial.println(recovered);
        Serial.print(F("[ecrecover] expected=0x")); Serial.println(F(ADDR_FROM));
        if (recovered.equalsIgnoreCase(ADDR_FROM)) {
            result = yp;
```

```
        }
    }
    return result;
}

uint8_t fetchNonce(uint64_t* nonce) {
    static const char requestPrefix[] =
        "{\"jsonrpc\":\"2.0\",\"id\":2,\"method\":\"eth_getTransactionCount\","
        "\"params\":[\"0x";
    static const char requestSuffix[] = "\",\"pending\"]}";
    const int bodyLen = (int)(sizeof(requestPrefix)-1) + 40 + (int)(sizeof(requestSuffix)-1);

    uint8_t result = 1U;
    for (uint8_t attempt = 0U; (attempt < TX_MAX_RETRIES) && (result != 0U); attempt++) {
        if (attempt != 0U) {
            delay(1000U);
        }
        if (!ensureWiFi()) {
            Serial.println(F("fetchNonce: WiFi reconnect failed"));
            continue;
        }
        WiFiSSLClient wifiClient;
#ifndef WIFI_DISABLE_CA_PINNING
        /* M-04: pin the RPC server's CA so a network MITM cannot serve a
         * forged certificate and feed crafted nonces/gas prices.
         * Define WIFI_DISABLE_CA_PINNING in config.h to skip this -- DEV ONLY,
         * leaves the connection vulnerable to MITM. */
        wifiClient.setCACert(WIFI_CA_CERT);
#endif
        HttpClient client(wifiClient, RPC_HOST, RPC_PORT);
        client.beginRequest();
        Serial.print(F("fetchNonce: connecting to "));
        Serial.print(F(RPC_HOST));
        Serial.println(F(RPC_PATH));
        int err = client.post(RPC_PATH);
        if (err != HTTP_SUCCESS) {
            Serial.print(F("fetchNonce: POST failed, err="));
            Serial.println(err);
            client.stop();
            continue;
        }
        client.sendHeader("Content-Type", "application/json");
        client.sendHeader("Content-Length", bodyLen);
#if defined(RPC_PROJECT_ID) && defined(RPC_API_SECRET)
        {
            char authBuf[AUTH_HEADER_BUF_SIZE];
            buildBasicAuthHeader(authBuf, sizeof(authBuf));
            client.sendHeader("Authorization", authBuf);
        }
#endif
        client.beginBody();
        client.print(requestPrefix);
        client.print(ADDR_FROM);
        client.print(requestSuffix);
        client.endRequest();

        int status = client.responseStatusCode();
        String resp = client.responseBody();
        client.stop();
        if (status == HTTP_OK) {
            int ri = resp.indexOf("\"result\"");
            int xi = (ri >= 0) ? resp.indexOf("0x", ri) : -1;
            if (xi >= 0) {
                uint64_t parsed = 0U;
                /* H-06: cap at 16 hex digits (uint64 capacity). A malicious or
                 * malformed RPC returning a longer hex string would silently
                 * overflow the uint64 and wrap to a small (replayed) nonce. */
                int digitCount = 0;
                bool overflowed = false;
                for (int i = xi + 2; i < (int)resp.length(); i++) {
                    char c = resp[i];
                    if (!((c>='0'&&c<='9')||(c>='a'&&c<='f')||(c>='A'&&c<='F'))) break;
                    if (digitCount >= 16) { overflowed = true; break; }
                    parsed = (parsed « 4) | fromHex(c);
                    digitCount++;
                }
                /* NEW-4: require >= 1 hex digit so "0x" / "0xZZZ" is not
                 * treated as a valid nonce=0 result on parse failure. */
                if (!overflowed && (digitCount > 0)) {
                    *nonce = parsed;
                    result = 0U;
                }
            }
        }
    }
    return result;
```

```
}

void setup() {
    Serial.begin(115200);
    delay(2000);

    /* Init SPI and PN532 */
    SPI.begin();
    if (!wallet.begin()) {
        Serial.println(F("PN532 init failed! Halting."));
        while(1);
    }
    Serial.println(F("PN532 OK"));

#ifdef WIFI_DISABLE_CA_PINNING
    Serial.println(F("  WIFI_DISABLE_CA_PINNING is set -- TLS certificate is NOT validated."));
    Serial.println(F("    Connection is vulnerable to MITM. DEV ONLY, do not use in production."));
#endif

    /* Connect to WiFi */
    Serial.print(F("Connecting to WiFi"));
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    uint8_t retries = 20U;
    while ((retries > 0U) && (WiFi.status() != WL_CONNECTED)) {
        retries--;
        delay(500U);
        Serial.print(F("."));
    }
    Serial.println();
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println(F("WiFi failed!"));
        while(1);
    }
    delay(2000); /* Allow network stack to stabilise before first SSL connection */

    /* Build ERC-20 calldata */
    uint8_t calldata[68];
    size_t calLen = encodeERC20Transfer(calldata);

    /* Build unsigned EIP-1559 transaction */
    Tx2 tx2;
    uint64_t fetchedNonce = 0U;
    if (fetchNonce(&fetchedNonce) != 0U) {
        Serial.println(F("fetchNonce failed! Halting."));
        while(1);
    }
    tx2.nonce              = fetchedNonce;
    tx2.maxPriorityFeePerGas = MAX_PRIORITY_FEE;
    tx2.maxFeePerGas       = MAX_FEE;
    tx2.gasLimit           = GAS_LIMIT_ERC20;
    tx2.to                 = ADDR_USDC;
    tx2.value              = 0;
    tx2.data               = calldata;
    tx2.dataLen            = calLen;
    tx2.chainId            = CHAIN_ID_SEPOLIA;

    /* RLP encode unsigned tx */
    static const size_t kRlpBufSize = 512U;
    uint8_t rlpUnsigned[kRlpBufSize];
    size_t rlpLen = rlpEncodeUnsignedTx(tx2, rlpUnsigned, kRlpBufSize);
    /* L-02: defense-in-depth. A USDC transfer EIP-1559 tx is ~150-200 bytes,
     * so 512 is plenty for the current shape -- but if a future change adds
     * an access list, auth list (EIP-7702), or larger tx.data, a silent
     * stack overflow would corrupt the return address. Halt explicitly. */
    /* rlpFinalize returns 0 on overflow (via safe_memcpy bounds check). The
     * post-check stays as defense-in-depth in case a future change skips
     * rlpFinalize's gate. */
    if ((rlpLen == 0U) || (rlpLen > kRlpBufSize)) {
        Serial.print(F("[fatal] rlpUnsigned overflow or empty: ")); Serial.println(rlpLen);
        while (true) {}
    }

    /* Keccak-256 hash */
    uint8_t hashKeccak[32];
    keccak256(static_cast<const uint8_t*>(rlpUnsigned), rlpLen, hashKeccak);
    printHex("[HASH]", hashKeccak, 32);

    /* BIP44 Ethereum path: m/44'/60'/0'/0/0 */
    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   */
    };

    /* === Sign with Cryptnox card over NFC (with retry on NFC dropout) === */
```

```
    Serial.println(F("Place Cryptnox card on PN532 reader..."));
    CW_SignResult signResult;
    for (uint8_t attempt = 0U; attempt < 3U; attempt++) {
        if (attempt != 0U) {
            delay(1000U);
        }
        CW_SecureSession session;
        if (!wallet.connect(session)) {
            continue;
        }
        Serial.println(F("Card connected, secure channel established."));

        CW_SignRequest signReq(session, CW_SIGN_DERIVE_K1, CW_SIGN_SIG_ECDSA_LOW_S, CW_SIGN_WITH_PIN);
        signReq.hash             = hashKeccak;
        signReq.hashLength       = static_cast<uint8_t>(CW_HASH_SIZE);
        signReq.derivePath       = eth_path;
        signReq.derivePathLength = static_cast<uint8_t>(sizeof(eth_path));
        (void)CW_Utils::safe_memcpy(signReq.pin, sizeof(signReq.pin),
                                    reinterpret_cast<const uint8_t*>(CARD_PIN), CARD_PIN_LEN);

        signResult = wallet.sign(signReq);
        wallet.disconnect(session);
        if (signResult.errorCode == CW_OK) {
            break;
        }
        if (signResult.errorCode == CW_SIGN_PIN_INCORRECT ||
            signResult.errorCode == CW_SIGN_NO_KEY_LOADED) {
            Serial.print(F("Card rejected sign command (error=0x"));
            Serial.print(signResult.errorCode, HEX);
            Serial.println(F(") -- check PIN and card initialisation. Halting."));
            /* M-05: explicitly wipe the PIN before halting. The CW_SignRequest
             * destructor would normally do this on scope exit, but the while(1)
             * below stays inside the for() body so the destructor never runs. */
            CW_Utils::secure_wipe(signReq.pin, sizeof(signReq.pin));
            while(1);
        }
        Serial.print(F("Sign attempt "));
        Serial.print(attempt + 1U);
        Serial.print(F(" failed, error=0x"));
        Serial.println(signResult.errorCode, HEX);
    }

    if (signResult.errorCode != CW_OK) {
        Serial.print(F("Sign failed: error=0x"));
        Serial.println(signResult.errorCode, HEX);
        while(1);
    }
    Serial.println(F("Signed."));

    const uint8_t* r = signResult.signature;       /* first 32 bytes */
    const uint8_t* s = signResult.signature + 32;  /* last  32 bytes */
    printHex("[SIG r]", r, 32);
    printHex("[SIG s]", s, 32);

    /* Determine yParity */
    uint8_t yParity = determineYParity(hashKeccak, r, s);
    if (yParity == YPARITY_UNKNOWN) {
        Serial.println(F("yParity determination failed! Halting."));
        while(1);
    }
    Serial.print(F("yParity: "));
    Serial.println(yParity);

    /* RLP encode signed tx and send */
    uint8_t rlpSigned[kRlpBufSize];
    size_t rlpSignedLen = rlpEncodeSignedTx(tx2, r, s, &yParity, rlpSigned, kRlpBufSize);
    /* L-02 + safe_memcpy gate inside rlpFinalize. */
    if ((rlpSignedLen == 0U) || (rlpSignedLen > kRlpBufSize)) {
        Serial.print(F("[fatal] rlpSigned overflow or empty: ")); Serial.println(rlpSignedLen);
        while (true) {}
    }

    Serial.println(F("Sending..."));
    if (sendRawTx(rlpSigned, rlpSignedLen)) {
        Serial.println(F("Transaction sent successfully!"));
    } else {
        Serial.println(F("Transaction FAILED."));
    }
}

void loop() {}
```

7.5 VerifyPin.ino

Minimal Cryptnox example: open a secure channel and submit a PIN.
Minimal Cryptnox example: open a secure channel and submit a PIN.Wiring & prerequisites:

- PN532 NFC reader on SPI, with SS on pin PN532_SS_PIN.
- A Cryptnox card initialised with a known PIN (use the Cryptnox CLI: `cryptnox init`, then `cryptnox seed generate`).
- DEMO_PIN must match the PIN set on the card.

What the sketch does in each loop iteration:

1. Connect to the card and establish the secure channel.
2. Submit DEMO_PIN over the secure channel.
3. Print "PIN accepted" if the card returned SW 0x9000, otherwise "PIN rejected (or channel error)". When the SW is not 0x9000 the SDK also prints the raw bytes (e.g. "Secured APDU: bad SW 0x63C2" means wrong PIN with 2 retries remaining).
4. Disconnect.

Warning

Every wrong PIN attempt decrements the card's retry counter. Reaching 0 retries permanently blocks the PIN; recovery then requires the PUK via `cryptnox` unblock_pin. Halt the sketch as soon as CryptnoxWallet::verifyPin returns `false` and check DEMO_PIN before retrying.

Note

The PIN "000000000" used by the project examples is a demo placeholder. In real deployments set a strong PIN, never commit source files containing a real PIN, and keep this value out of any version-controlled config.

```
/*
 * SPDX-License-Identifier: LGPL-3.0-or-later
 * Copyright (c) 2026 Cryptnox SA
 */

#include <CryptnoxWallet.h>
#include <SPI.h>

#define PN532_SS_PIN  (10U)

#define DEMO_PIN      "000000000"

ArduinoLoggerAdapter   serialAdapter;

PN532Adapter           nfc(serialAdapter, PN532_SS_PIN, &SPI);

ArduinoCryptoProvider  cryptoProvider;

ArduinoPlatform        platform;

CryptnoxWallet         wallet(nfc, serialAdapter, cryptoProvider, platform);

void setup() {
    serialAdapter.begin(115200);
    delay(1000);                              /* Arduino R4: wait for Serial */
    SPI.begin();

    if (!wallet.begin()) {
        serialAdapter.println(F("PN532 init failed"));
        while (1);
    }
}

void loop() {
    CW_SecureSession session;
    if (!wallet.connect(session)) {
        serialAdapter.println(F("Card not detected"));
        wallet.disconnect(session);
```

```
        delay(1000);
        return;
    }

    if (!wallet.verifyPin(session,
                          reinterpret_cast<const uint8_t*>(DEMO_PIN),
                          (uint8_t)strlen(DEMO_PIN))) {
        /* CRITICAL: do NOT loop on failure -- each wrong PIN attempt
         * decrements the card's retry counter (typically 3-5 tries) and
         * permanently blocks the PIN at zero. Halt and let the developer
         * inspect the Serial output / fix DEMO_PIN before retrying. */
        serialAdapter.println(F("PIN rejected -- halting to protect retry counter"));
        wallet.disconnect(session);
        while (1);
    }

    serialAdapter.println(F("PIN accepted"));
    wallet.disconnect(session);
    delay(1000);
}
```