Cryptnox SDK for Python Manual

Release 1.0

June 20, 2026

Contents:

1 Cryptnox SDK Python Overview

`cryptnox_sdk_py` is a Python 3 library used to communicate with the **Cryptnox Smartcard Applet**. It provides a high-level API to manage Cryptnox Hardware Wallet Cards, including initialization, secure channel setup, seed management, and cryptographic signing.

1.1 Supported Hardware

- **Cryptnox smart cards**
- **Standard PC/SC smart card readers**: either USB NFC reader or a USB smart card reader

Get your cards and readers here: shop.cryptnox.com

1.2 Features

- Establish communication with Cryptnox smart cards
- Initialize and manage card lifecycle
- Secure channel authentication and pairing
- Seed generation and restoration (BIP32 / BIP39 compatibility)
- ECDSA secp256k1 signing for blockchain applications

1.3 Installation

1.3.1 From PyPI

```
pip install cryptnox_sdk_py
```

1.3.2 From source

```
git clone https://github.com/cryptnox/cryptnox-sdk-py.git
cd cryptnox-sdk-py
pip install .
```

1.3.3 Requirements

- Python 3.11 – 3.13.7
- PC/SC Smart Card service (`pcscd`) on Linux

On Linux, ensure the PC/SC service is running:

```
sudo systemctl start pcscd
sudo systemctl enable pcscd
```

1.4 Quick Usage Examples

1.4.1 1. Connect to a Cryptnox Card

```
import cryptnox_sdk_py
from cryptnox_sdk_py import exceptions

connection = None
try:
    connection = cryptnox_sdk_py.Connection(0)
    card = cryptnox_sdk_py.factory.get_card(connection)
    # Card is loaded and can be used
    print(f"Card serial number: {card.serial_number}")
except exceptions.ReaderException:
    print("Reader not found at index")
except exceptions.CryptnoxException as error:
    # Issue loading the card
    print(error)
finally:
    # Always close the connection when done
    if connection:
        connection.disconnect()
```

1.4.2 2. Test PIN code

In the PIN verification example below the card must be initialized before calling verify_pin.

```
import cryptnox_sdk_py
from cryptnox_sdk_py import exceptions

connection = None
try:
    # Connect to the Cryptnox card first
    connection = cryptnox_sdk_py.Connection(0)  # Connect to card at index 0
    card = cryptnox_sdk_py.factory.get_card(connection)

    # Once connected, verify the PIN
    pin_to_test = "1234"  # Example PIN
    card.verify_pin(pin_to_test)
    print("PIN verified successfully. Card is ready for operations.")
except exceptions.ReaderException:
    print("Reader not found at index")
except exceptions.CryptnoxException as error:
    print(f"Error loading card: {error}")
except exceptions.PinException:
    print("Invalid PIN code.")
except exceptions.DataValidationException:
    print("Invalid PIN length or PIN authentication disabled.")
except exceptions.SoftLock:
    print("Card is locked. Please power cycle the card.")
finally:
```

```
    # Always close the connection when done
    if connection:
        connection.disconnect()
```

1.4.3 3. Generate a new seed

In the example below the card must be initialized before generating a seed.

```
import binascii
import cryptnox_sdk_py
from cryptnox_sdk_py import exceptions

PIN = "1234"  # or "" if the card was opened via challenge-response

def main():
    connection = None
    try:
        connection = cryptnox_sdk_py.Connection(0)
        card = cryptnox_sdk_py.factory.get_card(connection)

        seed_uid = card.generate_seed(PIN)
        # seed_uid is of type bytes: display in hex for readability
        print("Seed (primary node m) UID:", binascii.hexlify(seed_uid).
↪decode())
    except exceptions.ReaderException:
        print("Reader not found at index")
    except exceptions.CryptnoxException as err:
        print(f"Error loading card: {err}")
    except exceptions.KeyAlreadyGenerated:
        print("A seed is already generated on this card.")
    except exceptions.KeyGenerationException as err:
        print(f"Failed to generate seed: {err}")
    finally:
        # Always close the connection when done
        if connection:
            connection.disconnect()

if __name__ == "__main__":
    main()
```

1.5 Documentation

Full API reference: https://cryptnox.github.io/cryptnox-sdk-py/

1.6 License

cryptnox-sdk-py 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

2 API Reference

2.1 cryptnox_sdk_py package

2.1.1 Subpackages

cryptnox_sdk_py.card package

Submodules

cryptnox_sdk_py.card.authenticity module

Module containing check for verification of genuineness of a card

cryptnox_sdk_py.card.authenticity.**origin**(*connection:* Connection, *debug: bool* = *False*) → *Origin*

Check the origin of the card, whether it's a genuine :param Connection connection: connection to use for the card :param bool debug: print debug messages

Returns
Whether the card on the connection is genuine

Return type
Origin

cryptnox_sdk_py.card.authenticity.**session_public_key**(*connection:* Connection, *debug: bool* = *False*) → str

Check if the card in the reader is genuine Cryptnox product

Parameters

- **connection** (Connection) – Connection to use for operation
- **debug** (*bool*) – Prints information about communication

Returns
Session public key to use opening secure channel

Return type
str

Raises
GenuineCheckException – The card is not genuine

cryptnox_sdk_py.card.authenticity.**manufacturer_certificate**(*connection:* Connection, *debug: bool* = *False*) → str

Get the manufacturer certificate from the card in connection.

Parameters

- **connection** (Connection) – Connection to use for operation
- **debug** (*bool*) – Prints information about communication

Returns
Manufacturer certificate read from the card in hexadecimal format

Return type
str

cryptnox_sdk_py.card.base module

Module for keeping information about the card attached to the reader

class cryptnox_sdk_py.card.base.**User**(*name*, *email*)

Bases: tuple

email

Alias for field number 1

name

Alias for field number 0

class cryptnox_sdk_py.card.base.**SignatureCheckResult**(*message*, *signature*)

Bases: tuple

message

Alias for field number 0

signature

Alias for field number 1

class cryptnox_sdk_py.card.base.**Base**(*connection: Connection*, *serial: int*, *applet_version: List[int]*, *data: List[int] = None*, *debug: bool = False*)

Bases: object

Object that contains information about the card that is in the reader.

Parameters

- **connection** (Connection) – Connection to use for card initialization
- **debug** (*bool*) – Show debug information to the user.

Variables

- **applet_version** (*List[int]*) – Version of the applet on the card.
- **serial_number** (*int*) – Serial number of card.
- **session_public_key** (*str*) – Public key of the session.
- **initialized** (*bool*) – The card has been initialized with secrets.

Raises
CardTypeException – The card in the reader is not a Cryptnox card

PUK_LENGTH = 15

pin_rule = '4-9 digits'

type = 66

user_data = <cryptnox_sdk_py.card.user_data.UserDataBase object>

custom_bits = <cryptnox_sdk_py.card.custom_bits.CustomBitsBase object>

__init__(*connection:* Connection, *serial: int*, *applet_version: List[int]*, *data: List[int] = None*, *debug: bool = False*)

applet_version: List[int]

serial_number: int

auth_type: *AuthType*

abstract property select_apdu: List[int]

Value to add to select command to select the applet on the card :rtype: List[int]

Type
return

abstract property puk_rule: str

Human readable PUK code rule

Returns
Human readable PUK code rule

Return type
str

property alive: bool

The connection to the card is established and the card hasn't been changed :rtype: bool

Type
return

abstractmethod change_pairing_key(*index: int*, *pairing_key: bytes*, *puk: str = ''*) → None

Set the pairing key of the card

Parameters

- **index** (*int*) – Index of the pairing key
- **pairing_key** (*bytes*) – Pairing key to set for the card
- **puk** (*str*) – PUK code of the card

Raises

- *DataValidationException* – input data is not valid
- *SecureChannelException* – operation not allowed
- *PukException* – PUK code is not valid

change_pin(*new_pin: str*) → None

Change the current pin code of the card to a new pin code.

The method will set the given pin code as the pin code of the card. For it to work the card first must be opened with the current pin code.

Requires

- PIN code or challenge-response validated

Parameters
new_pin (*str*) – The desired PIN code to be set for the card (4-9 digits).

change_puk(*current_puk: str*, *new_puk: str*) → None
Change the current PUK code of the card to a new PUK code.

Parameters

- **current_puk** (*str*) – The current PUK code of the card
- **new_puk** (*str*) – The desired PUK code to be set for the card

check_init() → None
Check if the initialization has been done on the card.

It can be useful to check if the card is initialized before doing anything else, like asking for pin code from the user.

Raises
InitializationException – The card is not initialized

abstractmethod derive(*key_type:* KeyType = *KeyType.K1*, *path: str* = '')
Derive key on path and make it the current key in the card

Requires

- PIN code or challenge-response validated
- Seed must exist

Parameters

- **key_type** (KeyType) – Key type to do derive on
- **path** (*str*) – Path on which to do derivation

abstractmethod dual_seed_public_key(*pin: str* = '') → bytes
Get the public key from the card for dual initialization of the cards

Requires

- PIN code or challenge-response validated

Parameters
pin (*str*) – PIN code of card if it was opened with a PIN check

Returns
Public key and signature that can be sent into the other card

Return type
bytes

Raises
DataException – The received data is invalid

abstractmethod dual_seed_load(*data: bytes*, *pin: str* = '') → None
Load public key and signature from the other card into the card to generate same seed.

Requires

- PIN code or challenge-response validated

Parameters

- **pin** (*str*) – PIN code of card if it was opened with a PIN check
- **data** (*bytes*) – Public key and signature of public key from the other card

abstract property extended_public_key: bool

Extended public key turned on :rtype: bool

Type
return

abstractmethod generate_random_number(*size: int*) → bytes

Generate random number on the car and return it.

Parameters
size (*int*) – Output data size in bytes (between 16 and 64, mod 4)

Returns
Random number generated by the chip

Return type
bytes

Raises
DataValidationException – size in not a number between 16 and 64 or is not divisible by 4

abstractmethod generate_seed(*pin: str = ''*) → bytes

Generate a seed directly on the card.

Requires

- PIN code or challenge-response validated

Parameters
pin (*str, optional*) – PIN code of the card. Can be empty if card is opened with challenge-response validation

Returns
Primary node “m” UID (hash of public key)

Return type
bytes

Raises

- *KeyGenerationException* – There was an issue with generating the key
- *KeyAlreadyGenerated* – The card already has a seed generated

abstractmethod get_public_key(*derivation:* Derivation, *key_type:* KeyType = *KeyType.K1*, *path: str = ''*, *compressed: bool = True*) → str

Get the public key from the card.

Requires

- PIN code or challenge-response validated, except for PIN-less path
- Seed must exist

Parameters

- **derivation** (Derivation) – Derivation to use.
- **key_type** (KeyType) – Key type to use
- **path** (*str*)
- **compressed** (*bool*) – The returned value is in compressed format.

Returns
The public key for the given path in hexadecimal string format

Return type
str

Raises

- ***DerivationSelectionException*** – Card is not initialized with seed
- ***ReadPublicKeyException*** – Invalid data received from card

abstractmethod get_public_key_extended(*key_type:* KeyType = *KeyType.K1*, *puk: str* = *''*) → str

Get the extended public key (xpub) from the card.

Requires

- PIN code or challenge-response validated
- Seed must exist
- XPUB capability must be enabled (or PUK provided to enable it)

Parameters

- **key_type** (KeyType) – Key type to use (default: KeyType.K1)
- **puk** (*str*) – Optional PUK code to enable XPUB capability

Returns
Extended public key in hexadecimal string format

Return type
str

Raises

- ***exceptions.SeedException*** – If no seed exists on the card
- ***exceptions.ReadPublicKeyException*** – If invalid data received from card

get_public_key_clear(*derivation: int*, *path: str* = *''*, *compressed: bool* = *True*) → bytes

Get the public key within clear channel

Parameters

- **derivation** – Derivation + KeyType (e.g., Derivation.CURRENT_KEY + KeyType.K1)
- **path** – Optional BIP path string like “m/44’/0’/0’”
- **compressed** – Whether to return compressed format

Returns
Public key in bytes format

Raises

exceptions.ReadPublicKeyException – If bad data received

abstractmethod set_pubexport(*status: bool*, *p1: int*, *puk: str*) → None

Set pubexport capability (xpub or clear pubkey)

Parameters

- **status** – True to enable, False to disable
- **p1** – 0 for xpub, 1 for clear pubkey
- **puk** – PUK code

Raises

- *exceptions.DataValidationException* – If p1 is invalid
- *exceptions.PukException* – If PUK is incorrect

abstractmethod set_xpubread(*status: bool*, *puk: str*) → None

Set extended public key read capability

Parameters

- **status** – True to enable, False to disable
- **puk** – PUK code

abstractmethod set_clearpubkey(*status: bool*, *puk: str*) → None

Set clear public key read capability

Parameters

- **status** – True to enable, False to disable
- **puk** – PUK code

abstractmethod decrypt(*p1: int*, *pubkey: bytes*, *encrypted_data: bytes = b''*, *pin: str = ''*) → bytes

Decrypt data using ECIES (Elliptic Curve Integrated Encryption Scheme).

Generates a symmetric secret for simplified ECIES using an EC key in the BIP32 tree. This command is inspired by DECipher in OpenPGP smartcards. This allows asymmetric encryption using a key in the card seed tree. Anyone can encrypt with a public key from the card, and only the (private) key in the card can decrypt.

During the seed loading, the card saves in a dedicated key slot the result of a fixed derivation path. The child EC key used for this command is fixed (relative to a given seed). The path is computed with SLIP17 for the URI “openpgp://cryptnox” with index=0, and equals: m/17’/910196630’/2006168372’/332516148’/580566270’

The symmetric key is computed as SHA2(ECDH): SHA2-256(k.PubKey), where k is the private ECP256r1 key, the “decrypt” key slot.

Parameters

- **p1** (*int*) – 0x00 = Output symmetric key, 0x01 = Decrypt data in card
- **pubkey** (*bytes*) – Third party public key in X9.62 uncompressed format (0x04|X|Y, 65 bytes)

- **encrypted_data** (*bytes*) – Encrypted data (required when p1=1, must be 16-byte aligned)
- **pin** (*str*) – PIN code (required if no user key auth, right-padded with 0x00)

Returns
Symmetric key (p1=0) or decrypted data (p1=1)

Return type
bytes

Raises

- *exceptions.SeedException* – If no seed/key loaded
- *exceptions.PinException* – If PIN is incorrect
- *exceptions.DataValidationException* – If data length is incorrect
- *exceptions.GenericException* – If other errors occur

history(*index: int = 0*) → NamedTuple

Get history of hashes the card has signed regardless of any parameters given to sign

Requires

- PIN code or challenge-response validated

Parameters
index (*int*) – Index of entry in history

Returns
Return entry containing signing_counter, representing index of sign call, and hashed_data, the data that was signed

Return type
NamedTuple

property info: Dict[str, Any]

Get relevant information about the card.

Returns
Dictionary containing information for the card

Return type
Dict[str, Any]

init(*name: str*, *email: str*, *pin: str*, *puk: str*, *pairing_secret: bytes = BASIC_PAIRING_SECRET*, *nfc_sign: bool = False*) → bytes

Initialize the Cryptnox card.

Initialize the Cryptnox card with the owners name and email address. Set the PIN and PUK codes for authenticating with the card to be able to use it.

Parameters

- **name** (*str*) – Name of the card owner
- **email** (*str*) – Email of the card owner
- **pin** (*str*) – PIN code that will be used to open the card
- **puk** (*str*) – PUK code that will be used to open the card

- **pairing_secret** (*bytes*) – Pairing secret to use with the card
- **nfc_sign** (*bool*) – Signature command can be used over NFC, only available on certain type

Returns
Pairing secret

Return type
bytes

Raises
InitializationException – There was an issue with initialization

abstract property initialized: bool

Whether the card is initialized :rtype: bool

Type
return

abstractmethod load_seed(*seed: bytes*, *pin: str = ''*) → None

Load the given seed into the Cryptnox card.

Requires

- PIN code or challenge-response validated

Parameters

- **seed** (*bytes*) – Seed to initialize the card with
- **pin** (*str, optional*) – PIN code of the card. Can be empty if card is opened with challenge-response validation

Raises
KeyGenerationException – Data is not correct

property open: bool

Check if the user has authenticated.

Returns
Whether the user has authenticated using the PIN code or challenge-response validation

Return type
bool

property origin: *Origin*

Check the card origin.

Returns
Return if the card is original Cryptnox card, fake or there's an issue getting the information

Return type
Origin

abstract property pin_authentication: bool

Whether the PIN code can be used for authentication :rtype: bool

Type
return

abstract property pinless_enabled: bool
Return whether the card has a pinless path :rtype: bool

Type
return

abstractmethod reset(*puk: str*) → None
Reset the card and return it to factory settings.

Parameters
puk – PUK code associated with the card

abstract property seed_source: *SeedSource*
How the seed was generated :rtype: SeedSource

Type
return

abstractmethod set_pin_authentication(*status: bool*, *puk: str*) → None
Turn on/off authentication with the PIN code. Other methods can still be used.

Parameters

- **status** (*bool*) – Status of PIN authentication
- **puk** (*str*) – PUK code associated with the card

Raises

- *DataValidationException* – input data is not valid
- *PukException* – PUK code is not valid

abstractmethod set_pinless_path(*path: str*, *puk: str*) → None
Enable working with the card without a PIN on path.

Parameters

- **path** (*str*) – Path to be available without a PIN code
- **puk** (*str*) – PUK code of the card

Raises

- *DataValidationException* – input data is not valid
- *PukException* – PUK code is not valid

abstractmethod set_extended_public_key(*status: bool*, *puk: str*) → None
Turn on/off extended public key output.

Requires

- Seed must be loaded

Parameters

- **status** (*bool*) – Status of PIN authentication
- **puk** (*str*) – PUK code associated with the card

Raises

- ***DataValidationException*** – input data is not valid
- ***PukException*** – PUK code is not valid
- **KeyException** – Seed not found

abstractmethod sign(*data: bytes*, *derivation:* Derivation, *key_type:* KeyType = *KeyType.K1*, *path: str* = '', *pin: str* = '', *filter_eos: bool* = *False*) → bytes

Sign the message using given derivation.

Requires

- PIN code provided, authenticate with user key by signing same message or PIN-less path used
- Seed must be loaded

Parameters

- **data** (*bytes*) – Data to sign
- **derivation** (Derivation) – Derivation to use.
- **key_type** (KeyType, *optional*) – Key type to use. Defaults to K1
- **path** (*str, optional*) – Path of the key. If empty use main key
- **pin** (*str, optional*) – PIN code of the card
- **filter_eos** (*str, optional*) – Filter signature so it is valid for EOS network, might take longer. Defaults to False

Returns
The signature generated by the card in DER common format.

Return type
bytes

Raises
DataException – Invalid data received during signature

signature_check(*nonce: bytes*) → *SignatureCheckResult*

Sign random 32 bytes for validation that private key of public key is on the card.

This call doesn't increase signature counter and doesn't go into signature history.

Parameters
nonce (*bytes*) – random 16 bytes that will be used to sign

Returns
Message that was signed and the signature

Return type
SignatureCheckResult

Raises

- ***DataValidationException*** – Nonce has to be 16 bytes
- ***SeedException*** – There is no seed on the card
- ***DataException*** – Data returned from the card is not valid

abstract property signing_counter: int

Counter of how many times the card has been used to sign :rtype: int

Type
return

unblock_pin(*puk: str*, *new_pin: str*) → None

Verifies the user using the PUK code and sets a new PIN code on the card.

Method should be used when the user has forgotten this/hers PIN code. By entering the PUK code the user verifies his/hers identity and can set the new PIN code on the card. Can be used only if the card is locked.

Requires

- User PIN must be locked
- PIN code authentication must be enabled

Parameters

- **puk** (*str*) – PUK code for verification of the user, before changing the PIN code.
- **new_pin** (*str*) – The desired PIN code to be set for the card (4-9 digits).

Raises

- ***PukException*** – PUK code not valid
- ***CardNotBlocked*** – Card is not blocked, operation can't be done

abstractmethod user_key_add(*slot: SlotIndex*, *data_info: str*, *public_key: bytes*, *puk_code: str*, *cred_id: bytes = b''*) → None

Add user public key into the card for user authentication

Parameters

- **slot** (*int*) – Slot to write the public key to 1 - EC256R1 2 - RSA key, 2048 bits, public exponent must be 65537 3 - FIDO key
- **data_info** (*bytes*) – 64 bytes of user data
- **public_key** (*bytes*) – Public key of the secure element to be used for authentication
- **puk_code** (*str*) – PUK code of the card
- **cred_id** (*bytes, optional*) – Cred id. Used for FIDO2 authentication

Raises
DataValidationException – Invalid input data

abstractmethod user_key_delete(*slot: SlotIndex*, *puk_code: str*) → None

Delete the user key from slot and free up for insertion

Parameters

- **slot** (SlotIndex) – Slot to remove the key from
- **puk_code** (*str*) – PUK code of the card

Raises
DataValidationException – Invalid input data

abstractmethod user_key_info(*slot:* SlotIndex) → Tuple[str, str]

Get the description and public key of the user key

Requires

- PIN code or challenge-response validated

Parameters

slot (SlotIndex) – Index of slot for which to fetch the description

Returns

Description and public key in slot

Return type

tuple[str, str]

abstractmethod user_key_enabled(*slot_index:* SlotIndex) → bool

Check if user key is present in given slot

Parameters

slot_index (SlotIndex) – Slot index to check for

Returns

Whether the user key for slot is present

Return type

bool

abstractmethod user_key_challenge_response_nonce() → bytes

Get 32 bytes random value from the card that is used to open the card with a user key

Take nonce value from the card. Sign it with a third party application, like TPM. Send the signature back into the card using *user_key_challenge_response_open()*

Returns

32 bytes random value used as nonce

Return type

bytes

abstractmethod user_key_challenge_response_open(*slot:* SlotIndex, *signature: bytes*) → bool

Send the nonce signature to the card to open it for operations, like it was opened by a PIN code

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **signature** (*bytes*) – Signature generated by a third party like TPM.

Returns

Whether the challenge response authentication succeeded

Return type

bool

Raises

DataValidationException – invalid input data

abstractmethod user_key_signature_open(*slot:* SlotIndex, *message: bytes*, *signature: bytes*) → bool

Used for opening the card to sign the given message

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **message** (*bytes*) – Message that will be sent to sign operation
- **signature** (*bytes*) – Signature generated by a third party, like TPM, on the same message

Returns
Whether the challenge response authentication succeeded

Return type
bool

Raises
DataValidationException – invalid input data

abstract property valid_key: bool

Check if the card has a valid key

Returns
Whether the card has a valid key.

Return type
bool

static valid_pin(*pin: str*, *pin_name: str = 'pin'*) → str

Check if provided pin is valid

Parameters

- **pin** (*str*) – The pin to check if valid
- **pin_name** (*str*) – Value used in DataValidationException for pin name

Return str
Provided pin in str format if valid

Raises
DataValidationException – Provided pin is not valid

abstractmethod static valid_puk(*puk: str*, *puk_name: str = 'puk'*) → str

Check if provided puk is valid

Parameters

- **puk** (*str*) – The puk to check if valid (accepts all ASCII characters)
- **puk_name** (*str, optional*) – Value used in DataValidationException for puk name. Defaults to: puk

Return str
Provided puk in str format if valid

Raises
DataValidationException – Provided puk is not valid (wrong length or non-ASCII characters)

abstractmethod verify_pin(*pin: str | None = None*) → int | None

Verify PIN code, or query remaining retries without decrementing the counter.

If `pin` is `None`, returns the number of PIN attempts remaining (safe read). If `pin` is provided, verifies it and opens the card for protected operations.

Parameters
pin (*str or None*) – PIN code, or `None` to query remaining retries.

Returns
Retries remaining when `pin` is `None`, otherwise `None`.

Return type
int or None

Raises

- ***PinException*** – Invalid PIN code
- ***DataValidationException*** – Invalid length or PIN code authentication disabled
- ***SoftLock*** – The card has been locked and needs power cycling before it can be used again

abstractmethod get_manufacturer_certificate(*hexed: bool = True*) → Any

Get the manufacturer certificate from the card.

Parameters
hexed (*bool*) – Return the certificate in hexadecimal string format

Returns
Manufacturer certificate in hexadecimal string or bytes format

Return type
Any

cryptnox_sdk_py.card.basic_g1 module

Module containing class for Basic card of 1st generation

class cryptnox_sdk_py.card.basic_g1.**BasicG1**(**args*, ***kwargs*)

Bases: *Base*

Class containing functionality for Basic cards of the 1st generation

select_apdu = [160, 0, 0, 16, 0, 1, 18]

puk_rule = '12 ASCII characters'

PUK_LENGTH = 12

MAX_ASCII_LENGTH = 128

__init__(**args*, ***kwargs*)

change_pairing_key(*index: int*, *pairing_key: bytes*, *puk: str = ''*) → None

Set the pairing key of the card

Parameters

- **index** (*int*) – Index of the pairing key

- **pairing_key** (*bytes*) – Pairing key to set for the card
- **puk** (*str*) – PUK code of the card

Raises

- ***DataValidationException*** – input data is not valid
- ***SecureChannelException*** – operation not allowed
- ***PukException*** – PUK code is not valid

derive(*key_type:* KeyType = *KeyType.K1*, *path: str* = *''*)
Derive key on path and make it the current key in the card

Requires

- PIN code or challenge-response validated
- Seed must exist

Parameters

- **key_type** (KeyType) – Key type to do derive on
- **path** (*str*) – Path on which to do derivation

dual_seed_public_key(*pin: str* = *''*) → bytes
Get the public key from the card for dual initialization of the cards

Requires

- PIN code or challenge-response validated

Parameters
pin (*str*) – PIN code of card if it was opened with a PIN check

Returns
Public key and signature that can be sent into the other card

Return type
bytes

Raises
DataException – The received data is invalid

dual_seed_load(*data: bytes*, *pin: str* = *''*) → None
Load public key and signature from the other card into the card to generate same seed.

Requires

- PIN code or challenge-response validated

Parameters

- **pin** (*str*) – PIN code of card if it was opened with a PIN check
- **data** (*bytes*) – Public key and signature of public key from the other card

property extended_public_key: bool
Extended public key turned on :rtype: bool

Type
return

generate_random_number(*size: int*) → bytes

Generate random number on the car and return it.

Parameters
size (*int*) – Output data size in bytes (between 16 and 64, mod 4)

Returns
Random number generated by the chip

Return type
bytes

Raises
DataValidationException – size in not a number between 16 and 64 or is not divisible by 4

generate_seed(*pin: str = ''*) → bytes

Generate a seed directly on the card.

Requires

- PIN code or challenge-response validated

Parameters
pin (*str, optional*) – PIN code of the card. Can be empty if card is opened with challenge-response validation

Returns
Primary node “m” UID (hash of public key)

Return type
bytes

Raises

- *KeyGenerationException* – There was an issue with generating the key
- *KeyAlreadyGenerated* – The card already has a seed generated

get_manufacturer_certificate(*hexed: bool = True*)

Get the manufacturer certificate from the card.

Parameters
hexed (*bool*) – Return the certificate in hexadecimal string format

Returns
Manufacturer certificate in hexadecimal string or bytes format

Return type
Any

get_public_key(*derivation:* Derivation, *key_type:* KeyType = *KeyType.K1*, *path: str = ''*, *compressed: bool = True*, *hexed: bool = True*) → str

Get the public key from the card.

Requires

- PIN code or challenge-response validated, except for PIN-less path
- Seed must exist

Parameters

- **derivation** (Derivation) – Derivation to use.
- **key_type** (KeyType) – Key type to use
- **path** (*str*)
- **compressed** (*bool*) – The returned value is in compressed format.

Returns
The public key for the given path in hexadecimal string format

Return type
str

Raises

- ***DerivationSelectionException*** – Card is not initialized with seed
- ***ReadPublicKeyException*** – Invalid data received from card

get_public_key_extended(*key_type: KeyType = KeyType.K1*, *puk: str = ''*) → str
Get the extended public key (xpub) from the card.

Requires

- PIN code or challenge-response validated
- Seed must exist
- XPUB capability must be enabled (or PUK provided to enable it)

Parameters

- **key_type** (KeyType) – Key type to use (default: KeyType.K1)
- **puk** (*str*) – Optional PUK code to enable XPUB capability

Returns
Extended public key in hexadecimal string format

Return type
str

Raises

- ***exceptions.SeedException*** – If no seed exists on the card
- ***exceptions.ReadPublicKeyException*** – If invalid data received from card

get_public_key_clear(*derivation: int*, *path: str = ''*, *compressed: bool = True*) → bytes
Get the public key within clear channel

Parameters

- **derivation** – Derivation + KeyType (e.g., Derivation.CURRENT_KEY + KeyType.K1)
- **path** – Optional BIP path string like “m/44’/0’/0’”
- **compressed** – Whether to return compressed format

Returns
Public key in bytes format

Raises

exceptions.ReadPublicKeyException – If bad data received

decrypt(*p1: int*, *pubkey: bytes*, *encrypted_data: bytes = b''*, *pin: str = ''*) → bytes

Decrypt data using ECIES (Elliptic Curve Integrated Encryption Scheme).

Generates a symmetric secret for simplified ECIES using an EC key in the BIP32 tree. This command is inspired by DECipher in OpenPGP smartcards. This allows asymmetric encryption using a key in the card seed tree. Anyone can encrypt with a public key from the card, and only the (private) key in the card can decrypt.

During the seed loading, the card saves in a dedicated key slot the result of a fixed derivation path. The child EC key used for this command is fixed (relative to a given seed). The path is computed with SLIP17 for the URI "openpgp://cryptnox" with index=0, and equals: m/17'/910196630'/2006168372'/332516148'/580566270'

The symmetric key is computed as SHA2(ECDH): SHA2-256(k.PubKey), where k is the private ECP256r1 key, the "decrypt" key slot.

Parameters

- **p1** (*int*) – 0x00 = Output symmetric key, 0x01 = Decrypt data in card
- **pubkey** (*bytes*) – Third party public key in X9.62 uncompressed format (0x04|X|Y, 65 bytes)
- **encrypted_data** (*bytes*) – Encrypted data (required when p1=1, must be 16-byte aligned)
- **pin** (*str*) – PIN code (required if no user key auth, right-padded with 0x00)

Returns

Symmetric key (p1=0) or decrypted data (p1=1)

Return type

bytes

Raises

- *exceptions.SeedException* – If no seed/key loaded
- *exceptions.PinException* – If PIN is incorrect
- *exceptions.DataValidationException* – If data length is incorrect
- *exceptions.GenericException* – If other errors occur

history(*index: int = 0*) → NamedTuple

Get history of hashes the card has signed regardless of any parameters given to sign

Requires

- PIN code or challenge-response validated

Parameters

index (*int*) – Index of entry in history

Returns

Return entry containing signing_counter, representing index of sign call, and hashed_data, the data that was signed

Return type
NamedTuple

property initialized: bool

Whether the card is initialized :rtype: bool

Type
return

load_wrapped_seed(*seed: bytes*, *pin: str = ''*) → None

load_seed(*seed: bytes*, *pin: str = ''*) → None

Load the given seed into the Cryptnox card.

Requires

- PIN code or challenge-response validated

Parameters

- **seed** (*bytes*) – Seed to initialize the card with
- **pin** (*str, optional*) – PIN code of the card. Can be empty if card is opened with challenge-response validation

Raises
KeyGenerationException – Data is not correct

property pin_authentication: bool

Whether the PIN code can be used for authentication :rtype: bool

Type
return

property pinless_enabled: bool

Return whether the card has a pinless path :rtype: bool

Type
return

reset(*puk: str*) → None

Reset the card and return it to factory settings.

Parameters
puk – PUK code associated with the card

property seed_source: *SeedSource*

How the seed was generated :rtype: SeedSource

Type
return

set_pin_authentication(*status: bool*, *puk: str*) → None

Turn on/off authentication with the PIN code. Other methods can still be used.

Parameters

- **status** (*bool*) – Status of PIN authentication
- **puk** (*str*) – PUK code associated with the card

Raises

- *DataValidationException* – input data is not valid
- *PukException* – PUK code is not valid

set_pinless_path(*path: str*, *puk: str*) → None

Define a BIP-32 derivation path whose key can sign without PIN entry.

This is a deliberate design feature for payment/NFC tap-to-pay use cases where user interaction is not possible. The operation is PUK-protected: only the card owner who knows the PUK can enable or change the pinless path, limiting the attack surface to that single derivation path.

Parameters

- **path** (*str*) – BIP-32 path to allow pinless signing (empty to clear)
- **puk** (*str*) – PUK code authorising the change

Raises

- *SeedException* – No seed loaded on the card
- *PukException* – PUK is invalid
- *DataValidationException* – Path format is invalid

set_extended_public_key(*status: bool*, *puk: str*) → None

Set extended public key capability.

This is a convenience wrapper around set_pubexport(status, 0, puk). Use set_pubexport() directly for more control.

property signing_counter: int

Counter of how many times the card has been used to sign :rtype: int

Type
return

user_key_add(*slot:* SlotIndex, *data_info: str*, *public_key: bytes*, *puk_code: str*, *cred_id: bytes = b''*) → None

Add user public key into the card for user authentication

Parameters

- **slot** (*int*) – Slot to write the public key to 1 - EC256R1 2 - RSA key, 2048 bits, public exponent must be 65537 3 - FIDO key
- **data_info** (*bytes*) – 64 bytes of user data
- **public_key** (*bytes*) – Public key of the secure element to be used for authentication
- **puk_code** (*str*) – PUK code of the card
- **cred_id** (*bytes, optional*) – Cred id. Used for FIDO2 authentication

Raises
DataValidationException – Invalid input data

user_key_delete(*slot:* SlotIndex, *puk_code: str*) → None

Delete the user key from slot and free up for insertion

Parameters

- **slot** (SlotIndex) – Slot to remove the key from
- **puk_code** (*str*) – PUK code of the card

Raises

DataValidationException – Invalid input data

user_key_info(*slot:* SlotIndex) → Tuple[str, str]

Get the description and public key of the user key

Requires

- PIN code or challenge-response validated

Parameters

slot (SlotIndex) – Index of slot for which to fetch the description

Returns

Description and public key in slot

Return type

tuple[str, str]

user_key_enabled(*slot_index:* SlotIndex)

Check if user key is present in given slot

Parameters

slot_index (SlotIndex) – Slot index to check for

Returns

Whether the user key for slot is present

Return type

bool

user_key_challenge_response_nonce() → bytes

Get 32 bytes random value from the card that is used to open the card with a user key

Take nonce value from the card. Sign it with a third party application, like TPM. Send the signature back into the card using *user_key_challenge_response_open()*

Returns

32 bytes random value used as nonce

Return type

bytes

user_key_challenge_response_open(*slot:* SlotIndex, *signature: bytes*) → bool

Send the nonce signature to the card to open it for operations, like it was opened by a PIN code

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **signature** (*bytes*) – Signature generated by a third party like TPM.

Returns
Whether the challenge response authentication succeeded

Return type
bool

Raises
DataValidationException – invalid input data

user_key_signature_open(*slot:* SlotIndex, *message: bytes*, *signature: bytes*) → bool

Used for opening the card to sign the given message

Parameters

- **slot** (*SlotIndex*) – Slot to use to open the card
- **message** (*bytes*) – Message that will be sent to sign operation
- **signature** (*bytes*) – Signature generated by a third party, like TPM, on the same message

Returns
Whether the challenge response authentication succeeded

Return type
bool

Raises
DataValidationException – invalid input data

generate_seed_wrapper(*size: int = 2048*) → bytes

sign_public(*key_type:* KeyType = *KeyType.K1*) → bytes

sign(*data: bytes*, *derivation:* Derivation = *Derivation.CURRENT_KEY*, *key_type:* KeyType = *KeyType.K1*, *path: str = ''*, *pin: str = ''*, *filter_eos: bool = False*) → bytes

Sign the message using given derivation.

Requires

- PIN code provided, authenticate with user key by signing same message or PIN-less path used
- Seed must be loaded

Parameters

- **data** (*bytes*) – Data to sign
- **derivation** (*Derivation*) – Derivation to use.
- **key_type** (*KeyType, optional*) – Key type to use. Defaults to K1
- **path** (*str, optional*) – Path of the key. If empty use main key
- **pin** (*str, optional*) – PIN code of the card
- **filter_eos** (*str, optional*) – Filter signature so it is valid for EOS network, might take longer. Defaults to False

Returns
The signature generated by the card in DER common format.

Return type
bytes

Raises
DataException – Invalid data received during signature

property valid_key: bool

Check if the card has a valid key

Returns
Whether the card has a valid key.

Return type
bool

static valid_puk(*puk: str*, *puk_name: str = 'puk'*) → str

Check if provided puk is valid

Parameters

- **puk** (*str*) – The puk to check if valid (accepts all ASCII characters)
- **puk_name** (*str, optional*) – Value used in DataValidationException for puk name. Defaults to: puk

Return str
Provided puk in str format if valid

Raises
DataValidationException – Provided puk is not valid (wrong length or non-ASCII characters)

signature_check(*nonce: bytes*) → *SignatureCheckResult*

Sign random 32 bytes for validation that private key of public key is on the card.

BasicG1 cards do not support this operation. Use NFT card instead.

Raises
NotImplementedError – BasicG1 cards do not support signature_check

verify_pin(*pin: str | None = None*) → int | None

Verify PIN code, or query remaining retries without decrementing the counter.

If `pin` is `None`, returns the number of PIN attempts remaining (safe read). If `pin` is provided, verifies it and opens the card for protected operations.

Parameters
pin (*str or None*) – PIN code, or `None` to query remaining retries.

Returns
Retries remaining when `pin` is `None`, otherwise `None`.

Return type
int or None

Raises

- *PinException* – Invalid PIN code
- *DataValidationException* – Invalid length or PIN code authentication disabled

- *SoftLock* – The card has been locked and needs power cycling before it can be used again

set_pubexport(*status: bool*, *p1: int*, *puk: str*) → None
Set pubexport capability (xpub or clear pubkey)

Parameters

- **status** – True to enable, False to disable
- **p1** – 0 for xpub, 1 for clear pubkey
- **puk** – PUK code

Raises

- *exceptions.DataValidationException* – If p1 is invalid
- *exceptions.PukException* – If PUK is incorrect

set_xpubread(*status: bool*, *puk: str*) → None
Set extended public key read capability

Parameters

- **status** – True to enable, False to disable
- **puk** – PUK code

set_clearpubkey(*status: bool*, *puk: str*) → None
Set clear public key read capability

Parameters

- **status** – True to enable, False to disable
- **puk** – PUK code

cryptnox_sdk_py.card.custom_bits module

Module for making the uer data behave as a list

class cryptnox_sdk_py.card.custom_bits.**CustomBitsBase**
Bases: object

Class for User Data with all functions returning not implemented in case someone uses it on a card that doesn't support the feature

class cryptnox_sdk_py.card.custom_bits.**CustomBits**(*data*, *set_item_callback*)
Bases: object

__init__(*data*, *set_item_callback*)

cryptnox_sdk_py.card.nft module

Module containing class for NFT card

class cryptnox_sdk_py.card.nft.**Nft**(**args*, ***kwargs*)
Bases: *BasicG1*

Class containing functionality for NFT card which has limited capabilities

type = 78

__init__(**args*, ***kwargs*)

derive(*key_type:* KeyType = *KeyType.K1*, *path: str* = *''*)
Derive key on path and make it the current key in the card

Requires

- PIN code or challenge-response validated
- Seed must exist

Parameters

- **key_type** (KeyType) – Key type to do derive on
- **path** (*str*) – Path on which to do derivation

get_public_key(*derivation:* Derivation = *Derivation.CURRENT_KEY*, *key_type:* KeyType = *KeyType.K1*, *path: str* = *''*, *compressed: bool* = *False*, *hexed: bool* = *True*) → str
Get the public key from the card.

Requires

- PIN code or challenge-response validated, except for PIN-less path
- Seed must exist

Parameters

- **derivation** (Derivation) – Derivation to use.
- **key_type** (KeyType) – Key type to use
- **path** (*str*)
- **compressed** (*bool*) – The returned value is in compressed format.

Returns
The public key for the given path in hexadecimal string format

Return type
str

Raises

- ***DerivationSelectionException*** – Card is not initialized with seed
- ***ReadPublicKeyException*** – Invalid data received from card

get_public_key_clear(*derivation: int*, *path: str* = *''*, *compressed: bool* = *True*) → bytes
Get the public key within clear channel

Parameters

- **derivation** – Derivation + KeyType (e.g., Derivation.CURRENT_KEY + KeyType.K1)
- **path** – Optional BIP path string like “m/44’/0’/0’”
- **compressed** – Whether to return compressed format

Returns
Public key in bytes format

Raises
exceptions.ReadPublicKeyException – If bad data received

set_pubexport(*status: bool*, *p1: int*, *puk: str*) → None
Set pubexport capability (xpub or clear pubkey)

Parameters

- **status** – True to enable, False to disable
- **p1** – 0 for xpub, 1 for clear pubkey
- **puk** – PUK code

Raises

- *exceptions.DataValidationException* – If p1 is invalid
- *exceptions.PukException* – If PUK is incorrect

generate_random_number(*size: int*) → bytes
Generate random number on the car and return it.

Parameters
size (*int*) – Output data size in bytes (between 16 and 64, mod 4)

Returns
Random number generated by the chip

Return type
bytes

Raises
DataValidationException – size in not a number between 16 and 64 or is not divisible by 4

load_seed(*seed: bytes*, *pin: str = ''*) → None
Load the given seed into the Cryptnox card.

Requires

- PIN code or challenge-response validated

Parameters

- **seed** (*bytes*) – Seed to initialize the card with
- **pin** (*str, optional*) – PIN code of the card. Can be empty if card is opened with challenge-response validation

Raises
KeyGenerationException – Data is not correct

set_pin_authentication(*status: bool*, *puk: str*) → None
Turn on/off authentication with the PIN code. Other methods can still be used.

Parameters

- **status** (*bool*) – Status of PIN authentication

- **puk** (*str*) – PUK code associated with the card

Raises

- ***DataValidationException*** – input data is not valid
- ***PukException*** – PUK code is not valid

set_pinless_path(*path: str*, *puk: str*) → None

Define a BIP-32 derivation path whose key can sign without PIN entry.

This is a deliberate design feature for payment/NFC tap-to-pay use cases where user interaction is not possible. The operation is PUK-protected: only the card owner who knows the PUK can enable or change the pinless path, limiting the attack surface to that single derivation path.

Parameters

- **path** (*str*) – BIP-32 path to allow pinless signing (empty to clear)
- **puk** (*str*) – PUK code authorising the change

Raises

- ***SeedException*** – No seed loaded on the card
- ***PukException*** – PUK is invalid
- ***DataValidationException*** – Path format is invalid

user_key_add(*slot:* SlotIndex, *data_info: str*, *public_key: bytes*, *puk_code: str*, *cred_id: bytes = b''*) → None

Add user public key into the card for user authentication

Parameters

- **slot** (*int*) – Slot to write the public key to 1 - EC256R1 2 - RSA key, 2048 bits, public exponent must be 65537 3 - FIDO key
- **data_info** (*bytes*) – 64 bytes of user data
- **public_key** (*bytes*) – Public key of the secure element to be used for authentication
- **puk_code** (*str*) – PUK code of the card
- **cred_id** (*bytes, optional*) – Cred id. Used for FIDO2 authentication

Raises

DataValidationException – Invalid input data

user_key_delete(*slot:* SlotIndex, *puk_code: str*) → None

Delete the user key from slot and free up for insertion

Parameters

- **slot** (SlotIndex) – Slot to remove the key from
- **puk_code** (*str*) – PUK code of the card

Raises

DataValidationException – Invalid input data

user_key_info(*slot:* SlotIndex) → Tuple[str, str]

Get the description and public key of the user key

Requires

- PIN code or challenge-response validated

Parameters

slot (SlotIndex) – Index of slot for which to fetch the description

Returns

Description and public key in slot

Return type

tuple[str, str]

user_key_enabled(*slot_index:* SlotIndex)

Check if user key is present in given slot

Parameters

slot_index (SlotIndex) – Slot index to check for

Returns

Whether the user key for slot is present

Return type

bool

user_key_challenge_response_nonce() → bytes

Get 32 bytes random value from the card that is used to open the card with a user key

Take nonce value from the card. Sign it with a third party application, like TPM. Send the signature back into the card using *user_key_challenge_response_open()*

Returns

32 bytes random value used as nonce

Return type

bytes

user_key_challenge_response_open(*slot:* SlotIndex, *signature: bytes*) → bool

Send the nonce signature to the card to open it for operations, like it was opened by a PIN code

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **signature** (*bytes*) – Signature generated by a third party like TPM.

Returns

Whether the challenge response authentication succeeded

Return type

bool

Raises

DataValidationException – invalid input data

user_key_signature_open(*slot: SlotIndex*, *message: bytes*, *signature: bytes*) → bool

Used for opening the card to sign the given message

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **message** (*bytes*) – Message that will be sent to sign operation
- **signature** (*bytes*) – Signature generated by a third party, like TPM, on the same message

Returns

Whether the challenge response authentication succeeded

Return type

bool

Raises

DataValidationException – invalid input data

signature_check(*nonce: bytes*) → *SignatureCheckResult*

Sign random 32 bytes for validation that private key of public key is on the card.

BasicG1 cards do not support this operation. Use NFT card instead.

Raises

NotImplementedError – BasicG1 cards do not support signature_check

cryptnox_sdk_py.card.user_data module

Module for making the uer data behave as a list

class cryptnox_sdk_py.card.user_data.**UserDataBase**

Bases: object

Class for User Data with all functions returning not implemented in case someone uses it on a card that doesn't support the feature

class cryptnox_sdk_py.card.user_data.**UserData**(*card*, *slot_offset: int = 0*, *reading_index_offset: int = 0*)

Bases: object

User data that behaves as a list and can fetch different user slots from the card

__init__(*card*, *slot_offset: int = 0*, *reading_index_offset: int = 0*)

Module contents

Module that contains classes for various Cryptnox cards.

class cryptnox_sdk_py.card.**Base**(*connection: Connection*, *serial: int*, *applet_version: List[int]*, *data: List[int] = None*, *debug: bool = False*)

Bases: object

Object that contains information about the card that is in the reader.

Parameters

- **connection** (Connection) – Connection to use for card initialization

- **debug** (*bool*) – Show debug information to the user.

Variables

- **applet_version** (*List[int]*) – Version of the applet on the card.
- **serial_number** (*int*) – Serial number of card.
- **session_public_key** (*str*) – Public key of the session.
- **initialized** (*bool*) – The card has been initialized with secrets.

Raises

CardTypeException – The card in the reader is not a Cryptnox card

PUK_LENGTH = 15

pin_rule = '4-9 digits'

type = 66

user_data = <cryptnox_sdk_py.card.user_data.UserDataBase object>

custom_bits = <cryptnox_sdk_py.card.custom_bits.CustomBitsBase object>

__init__(*connection: Connection*, *serial: int*, *applet_version: List[int]*, *data: List[int] = None*, *debug: bool = False*)

applet_version: List[int]

serial_number: int

auth_type: *AuthType*

abstract property select_apdu: List[int]

Value to add to select command to select the applet on the card :rtype: List[int]

Type

return

abstract property puk_rule: str

Human readable PUK code rule

Returns

Human readable PUK code rule

Return type

str

property alive: bool

The connection to the card is established and the card hasn't been changed :rtype: bool

Type

return

abstractmethod change_pairing_key(*index: int*, *pairing_key: bytes*, *puk: str = ''*) → None

Set the pairing key of the card

Parameters

- **index** (*int*) – Index of the pairing key

- **pairing_key** (*bytes*) – Pairing key to set for the card
- **puk** (*str*) – PUK code of the card

Raises

- ***DataValidationException*** – input data is not valid
- ***SecureChannelException*** – operation not allowed
- ***PukException*** – PUK code is not valid

change_pin(*new_pin: str*) → None

Change the current pin code of the card to a new pin code.

The method will set the given pin code as the pin code of the card. For it to work the card first must be opened with the current pin code.

Requires

- PIN code or challenge-response validated

Parameters

new_pin (*str*) – The desired PIN code to be set for the card (4-9 digits).

change_puk(*current_puk: str*, *new_puk: str*) → None

Change the current PUK code of the card to a new PUK code.

Parameters

- **current_puk** (*str*) – The current PUK code of the card
- **new_puk** (*str*) – The desired PUK code to be set for the card

check_init() → None

Check if the initialization has been done on the card.

It can be useful to check if the card is initialized before doing anything else, like asking for pin code from the user.

Raises

InitializationException – The card is not initialized

abstractmethod derive(*key_type:* KeyType = *KeyType.K1*, *path: str = ''*)

Derive key on path and make it the current key in the card

Requires

- PIN code or challenge-response validated
- Seed must exist

Parameters

- **key_type** (KeyType) – Key type to do derive on
- **path** (*str*) – Path on which to do derivation

abstractmethod dual_seed_public_key(*pin: str = ''*) → bytes

Get the public key from the card for dual initialization of the cards

Requires

- PIN code or challenge-response validated

Parameters
pin (*str*) – PIN code of card if it was opened with a PIN check

Returns
Public key and signature that can be sent into the other card

Return type
bytes

Raises
DataException – The received data is invalid

abstractmethod dual_seed_load(*data: bytes*, *pin: str = ''*) → None
Load public key and signature from the other card into the card to generate same seed.

Requires

- PIN code or challenge-response validated

Parameters

- **pin** (*str*) – PIN code of card if it was opened with a PIN check
- **data** (*bytes*) – Public key and signature of public key from the other card

abstract property extended_public_key: bool
Extended public key turned on :rtype: bool

Type
return

abstractmethod generate_random_number(*size: int*) → bytes
Generate random number on the car and return it.

Parameters
size (*int*) – Output data size in bytes (between 16 and 64, mod 4)

Returns
Random number generated by the chip

Return type
bytes

Raises
DataValidationException – size in not a number between 16 and 64 or is not divisible by 4

abstractmethod generate_seed(*pin: str = ''*) → bytes
Generate a seed directly on the card.

Requires

- PIN code or challenge-response validated

Parameters
pin (*str, optional*) – PIN code of the card. Can be empty if card is opened with challenge-response validation

Returns
Primary node “m” UID (hash of public key)

Return type
bytes

Raises

- ***KeyGenerationException*** – There was an issue with generating the key
- ***KeyAlreadyGenerated*** – The card already has a seed generated

abstractmethod get_public_key(*derivation:* Derivation, *key_type:* KeyType = *KeyType.K1*, *path: str = ''*, *compressed: bool = True*) → str

Get the public key from the card.

Requires

- PIN code or challenge-response validated, except for PIN-less path
- Seed must exist

Parameters

- **derivation** (Derivation) – Derivation to use.
- **key_type** (KeyType) – Key type to use
- **path** (*str*)
- **compressed** (*bool*) – The returned value is in compressed format.

Returns
The public key for the given path in hexadecimal string format

Return type
str

Raises

- ***DerivationSelectionException*** – Card is not initialized with seed
- ***ReadPublicKeyException*** – Invalid data received from card

abstractmethod get_public_key_extended(*key_type:* KeyType = *KeyType.K1*, *puk: str = ''*) → str

Get the extended public key (xpub) from the card.

Requires

- PIN code or challenge-response validated
- Seed must exist
- XPUB capability must be enabled (or PUK provided to enable it)

Parameters

- **key_type** (KeyType) – Key type to use (default: KeyType.K1)
- **puk** (*str*) – Optional PUK code to enable XPUB capability

Returns
Extended public key in hexadecimal string format

Return type
str

Raises

- *exceptions.SeedException* – If no seed exists on the card
- *exceptions.ReadPublicKeyException* – If invalid data received from card

get_public_key_clear(*derivation: int*, *path: str = ''*, *compressed: bool = True*) → bytes

Get the public key within clear channel

Parameters

- **derivation** – Derivation + KeyType (e.g., Derivation.CURRENT_KEY + KeyType.K1)
- **path** – Optional BIP path string like "m/44'/0'/0'"
- **compressed** – Whether to return compressed format

Returns

Public key in bytes format

Raises

exceptions.ReadPublicKeyException – If bad data received

abstractmethod set_pubexport(*status: bool*, *p1: int*, *puk: str*) → None

Set pubexport capability (xpub or clear pubkey)

Parameters

- **status** – True to enable, False to disable
- **p1** – 0 for xpub, 1 for clear pubkey
- **puk** – PUK code

Raises

- *exceptions.DataValidationException* – If p1 is invalid
- *exceptions.PukException* – If PUK is incorrect

abstractmethod set_xpubread(*status: bool*, *puk: str*) → None

Set extended public key read capability

Parameters

- **status** – True to enable, False to disable
- **puk** – PUK code

abstractmethod set_clearpubkey(*status: bool*, *puk: str*) → None

Set clear public key read capability

Parameters

- **status** – True to enable, False to disable
- **puk** – PUK code

abstractmethod decrypt(*p1: int*, *pubkey: bytes*, *encrypted_data: bytes = b''*, *pin: str = ''*) → bytes

Decrypt data using ECIES (Elliptic Curve Integrated Encryption Scheme).

Generates a symmetric secret for simplified ECIES using an EC key in the BIP32 tree. This command is inspired by DECipher in OpenPGP smartcards. This allows asymmetric encryption using a key in the card seed tree. Anyone can encrypt with a public key from the card, and only the (private) key in the card can decrypt.

During the seed loading, the card saves in a dedicated key slot the result of a fixed derivation path. The child EC key used for this command is fixed (relative to a given seed). The path is computed with SLIP17 for the URI "openpgp://cryptnox" with index=0, and equals: m/17'/910196630'/2006168372'/332516148'/580566270'

The symmetric key is computed as SHA2(ECDH): SHA2-256(k.PubKey), where k is the private ECP256r1 key, the "decrypt" key slot.

Parameters

- **p1** (*int*) – 0x00 = Output symmetric key, 0x01 = Decrypt data in card
- **pubkey** (*bytes*) – Third party public key in X9.62 uncompressed format (0x04|X|Y, 65 bytes)
- **encrypted_data** (*bytes*) – Encrypted data (required when p1=1, must be 16-byte aligned)
- **pin** (*str*) – PIN code (required if no user key auth, right-padded with 0x00)

Returns
Symmetric key (p1=0) or decrypted data (p1=1)

Return type
bytes

Raises

- ***exceptions.SeedException*** – If no seed/key loaded
- ***exceptions.PinException*** – If PIN is incorrect
- ***exceptions.DataValidationException*** – If data length is incorrect
- ***exceptions.GenericException*** – If other errors occur

history(*index: int = 0*) → NamedTuple

Get history of hashes the card has signed regardless of any parameters given to sign

Requires

- PIN code or challenge-response validated

Parameters
index (*int*) – Index of entry in history

Returns
Return entry containing signing_counter, representing index of sign call, and hashed_data, the data that was signed

Return type
NamedTuple

property info: Dict[str, Any]

Get relevant information about the card.

Returns
Dictionary containing information for the card

Return type
Dict[str, Any]

init(*name: str*, *email: str*, *pin: str*, *puk: str*, *pairing_secret: bytes* = *BASIC_PAIRING_SECRET*, *nfc_sign: bool* = *False*) → bytes

Initialize the Cryptnox card.

Initialize the Cryptnox card with the owners name and email address. Set the PIN and PUK codes for authenticating with the card to be able to use it.

Parameters

- **name** (*str*) – Name of the card owner
- **email** (*str*) – Email of the card owner
- **pin** (*str*) – PIN code that will be used to open the card
- **puk** (*str*) – PUK code that will be used to open the card
- **pairing_secret** (*bytes*) – Pairing secret to use with the card
- **nfc_sign** (*bool*) – Signature command can be used over NFC, only available on certain type

Returns
Pairing secret

Return type
bytes

Raises
InitializationException – There was an issue with initialization

abstract property initialized: bool

Whether the card is initialized :rtype: bool

Type
return

abstractmethod load_seed(*seed: bytes*, *pin: str* = *''*) → None

Load the given seed into the Cryptnox card.

Requires

- PIN code or challenge-response validated

Parameters

- **seed** (*bytes*) – Seed to initialize the card with
- **pin** (*str, optional*) – PIN code of the card. Can be empty if card is opened with challenge-response validation

Raises
KeyGenerationException – Data is not correct

property open: bool

Check if the user has authenticated.

Returns
Whether the user has authenticated using the PIN code or challenge-response validation

Return type
bool

property origin: *Origin*

Check the card origin.

Returns
Return if the card is original Cryptnox card, fake or there's an issue getting the information

Return type
Origin

abstract property pin_authentication: bool

Whether the PIN code can be used for authentication :rtype: bool

Type
return

abstract property pinless_enabled: bool

Return whether the card has a pinless path :rtype: bool

Type
return

abstractmethod reset(*puk: str*) → None

Reset the card and return it to factory settings.

Parameters
puk – PUK code associated with the card

abstract property seed_source: *SeedSource*

How the seed was generated :rtype: SeedSource

Type
return

abstractmethod set_pin_authentication(*status: bool*, *puk: str*) → None

Turn on/off authentication with the PIN code. Other methods can still be used.

Parameters

- **status** (*bool*) – Status of PIN authentication
- **puk** (*str*) – PUK code associated with the card

Raises

- ***DataValidationException*** – input data is not valid
- ***PukException*** – PUK code is not valid

abstractmethod set_pinless_path(*path: str*, *puk: str*) → None
Enable working with the card without a PIN on path.

Parameters

- **path** (*str*) – Path to be available without a PIN code
- **puk** (*str*) – PUK code of the card

Raises

- ***DataValidationException*** – input data is not valid
- ***PukException*** – PUK code is not valid

abstractmethod set_extended_public_key(*status: bool*, *puk: str*) → None
Turn on/off extended public key output.

Requires

- Seed must be loaded

Parameters

- **status** (*bool*) – Status of PIN authentication
- **puk** (*str*) – PUK code associated with the card

Raises

- ***DataValidationException*** – input data is not valid
- ***PukException*** – PUK code is not valid
- **KeyException** – Seed not found

abstractmethod sign(*data: bytes*, *derivation:* Derivation, *key_type:* KeyType = *KeyType.K1*, *path: str = ''*, *pin: str = ''*, *filter_eos: bool = False*) → bytes
Sign the message using given derivation.

Requires

- PIN code provided, authenticate with user key by signing same message or PIN-less path used
- Seed must be loaded

Parameters

- **data** (*bytes*) – Data to sign
- **derivation** (Derivation) – Derivation to use.
- **key_type** (KeyType, *optional*) – Key type to use. Defaults to K1
- **path** (*str, optional*) – Path of the key. If empty use main key
- **pin** (*str, optional*) – PIN code of the card
- **filter_eos** (*str, optional*) – Filter signature so it is valid for EOS network, might take longer. Defaults to False

Returns
The signature generated by the card in DER common format.

Return type
bytes

Raises
DataException – Invalid data received during signature

signature_check(*nonce: bytes*) → *SignatureCheckResult*

Sign random 32 bytes for validation that private key of public key is on the card.

This call doesn't increase signature counter and doesn't go into signature history.

Parameters
nonce (*bytes*) – random 16 bytes that will be used to sign

Returns
Message that was signed and the signature

Return type
SignatureCheckResult

Raises

- *DataValidationException* – Nonce has to be 16 bytes
- *SeedException* – There is no seed on the card
- *DataException* – Data returned from the card is not valid

abstract property signing_counter: int

Counter of how many times the card has been used to sign :rtype: int

Type
return

unblock_pin(*puk: str*, *new_pin: str*) → None

Verifies the user using the PUK code and sets a new PIN code on the card.

Method should be used when the user has forgotten this/hers PIN code. By entering the PUK code the user verifies his/hers identity and can set the new PIN code on the card. Can be used only if the card is locked.

Requires

- User PIN must be locked
- PIN code authentication must be enabled

Parameters

- **puk** (*str*) – PUK code for verification of the user, before changing the PIN code.
- **new_pin** (*str*) – The desired PIN code to be set for the card (4-9 digits).

Raises

- *PukException* – PUK code not valid
- *CardNotBlocked* – Card is not blocked, operation can't be done

abstractmethod user_key_add(*slot:* SlotIndex, *data_info: str*, *public_key: bytes*, *puk_code: str*, *cred_id: bytes = b''*) → None

Add user public key into the card for user authentication

Parameters

- **slot** (*int*) – Slot to write the public key to 1 - EC256R1 2 - RSA key, 2048 bits, public exponent must be 65537 3 - FIDO key
- **data_info** (*bytes*) – 64 bytes of user data
- **public_key** (*bytes*) – Public key of the secure element to be used for authentication
- **puk_code** (*str*) – PUK code of the card
- **cred_id** (*bytes, optional*) – Cred id. Used for FIDO2 authentication

Raises

DataValidationException – Invalid input data

abstractmethod user_key_delete(*slot:* SlotIndex, *puk_code: str*) → None

Delete the user key from slot and free up for insertion

Parameters

- **slot** (SlotIndex) – Slot to remove the key from
- **puk_code** (*str*) – PUK code of the card

Raises

DataValidationException – Invalid input data

abstractmethod user_key_info(*slot:* SlotIndex) → Tuple[str, str]

Get the description and public key of the user key

Requires

- PIN code or challenge-response validated

Parameters

slot (SlotIndex) – Index of slot for which to fetch the description

Returns

Description and public key in slot

Return type

tuple[str, str]

abstractmethod user_key_enabled(*slot_index:* SlotIndex) → bool

Check if user key is present in given slot

Parameters

slot_index (SlotIndex) – Slot index to check for

Returns

Whether the user key for slot is present

Return type

bool

abstractmethod user_key_challenge_response_nonce() → bytes

Get 32 bytes random value from the card that is used to open the card with a user key

Take nonce value from the card. Sign it with a third party application, like TPM. Send the signature back into the card using *user_key_challenge_response_open()*

Returns
32 bytes random value used as nonce

Return type
bytes

abstractmethod user_key_challenge_response_open(*slot:* SlotIndex, *signature: bytes*) → bool

Send the nonce signature to the card to open it for operations, like it was opened by a PIN code

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **signature** (*bytes*) – Signature generated by a third party like TPM.

Returns
Whether the challenge response authentication succeeded

Return type
bool

Raises
DataValidationException – invalid input data

abstractmethod user_key_signature_open(*slot:* SlotIndex, *message: bytes*, *signature: bytes*) → bool

Used for opening the card to sign the given message

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **message** (*bytes*) – Message that will be sent to sign operation
- **signature** (*bytes*) – Signature generated by a third party, like TPM, on the same message

Returns
Whether the challenge response authentication succeeded

Return type
bool

Raises
DataValidationException – invalid input data

abstract property valid_key: bool

Check if the card has a valid key

Returns
Whether the card has a valid key.

Return type
bool

static valid_pin(*pin: str*, *pin_name: str = 'pin'*) → str

Check if provided pin is valid

Parameters

- **pin** (*str*) – The pin to check if valid
- **pin_name** (*str*) – Value used in DataValidationException for pin name

Return str
Provided pin in str format if valid

Raises
DataValidationException – Provided pin is not valid

abstractmethod static valid_puk(*puk: str*, *puk_name: str = 'puk'*) → str

Check if provided puk is valid

Parameters

- **puk** (*str*) – The puk to check if valid (accepts all ASCII characters)
- **puk_name** (*str, optional*) – Value used in DataValidationException for puk name. Defaults to: puk

Return str
Provided puk in str format if valid

Raises
DataValidationException – Provided puk is not valid (wrong length or non-ASCII characters)

abstractmethod verify_pin(*pin: str | None = None*) → int | None

Verify PIN code, or query remaining retries without decrementing the counter.

If `pin` is `None`, returns the number of PIN attempts remaining (safe read). If `pin` is provided, verifies it and opens the card for protected operations.

Parameters
pin (*str or None*) – PIN code, or `None` to query remaining retries.

Returns
Retries remaining when `pin` is `None`, otherwise `None`.

Return type
int or None

Raises

- *PinException* – Invalid PIN code
- *DataValidationException* – Invalid length or PIN code authentication disabled
- *SoftLock* – The card has been locked and needs power cycling before it can be used again

abstractmethod get_manufacturer_certificate(*hexed: bool = True*) → Any

Get the manufacturer certificate from the card.

Parameters
hexed (*bool*) – Return the certificate in hexadecimal string format

Returns
Manufacturer certificate in hexadecimal string or bytes format

Return type
Any

class cryptnox_sdk_py.card.**BasicG1**(**args*, ***kwargs*)
Bases: *Base*

Class containing functionality for Basic cards of the 1st generation

select_apdu = [160, 0, 0, 16, 0, 1, 18]

puk_rule = '12 ASCII characters'

PUK_LENGTH = 12

MAX_ASCII_LENGTH = 128

__init__(**args*, ***kwargs*)

change_pairing_key(*index: int*, *pairing_key: bytes*, *puk: str = ''*) → None
Set the pairing key of the card

Parameters

- **index** (*int*) – Index of the pairing key
- **pairing_key** (*bytes*) – Pairing key to set for the card
- **puk** (*str*) – PUK code of the card

Raises

- ***DataValidationException*** – input data is not valid
- ***SecureChannelException*** – operation not allowed
- ***PukException*** – PUK code is not valid

derive(*key_type:* KeyType *= KeyType.K1*, *path: str = ''*)
Derive key on path and make it the current key in the card

Requires

- PIN code or challenge-response validated
- Seed must exist

Parameters

- **key_type** (KeyType) – Key type to do derive on
- **path** (*str*) – Path on which to do derivation

dual_seed_public_key(*pin: str = ''*) → bytes
Get the public key from the card for dual initialization of the cards

Requires

- PIN code or challenge-response validated

Parameters
pin (*str*) – PIN code of card if it was opened with a PIN check

Returns
Public key and signature that can be sent into the other card

Return type
bytes

Raises
DataException – The received data is invalid

dual_seed_load(*data: bytes*, *pin: str = ''*) → None
Load public key and signature from the other card into the card to generate same seed.

Requires

- PIN code or challenge-response validated

Parameters

- **pin** (*str*) – PIN code of card if it was opened with a PIN check
- **data** (*bytes*) – Public key and signature of public key from the other card

property extended_public_key: bool
Extended public key turned on :rtype: bool

Type
return

generate_random_number(*size: int*) → bytes
Generate random number on the car and return it.

Parameters
size (*int*) – Output data size in bytes (between 16 and 64, mod 4)

Returns
Random number generated by the chip

Return type
bytes

Raises
DataValidationException – size in not a number between 16 and 64 or is not divisible by 4

generate_seed(*pin: str = ''*) → bytes
Generate a seed directly on the card.

Requires

- PIN code or challenge-response validated

Parameters
pin (*str, optional*) – PIN code of the card. Can be empty if card is opened with challenge-response validation

Returns
Primary node “m” UID (hash of public key)

Return type
bytes

Raises

- *KeyGenerationException* – There was an issue with generating the key
- *KeyAlreadyGenerated* – The card already has a seed generated

get_manufacturer_certificate(*hexed: bool = True*)

Get the manufacturer certificate from the card.

Parameters

hexed (*bool*) – Return the certificate in hexadecimal string format

Returns

Manufacturer certificate in hexadecimal string or bytes format

Return type

Any

get_public_key(*derivation:* Derivation, *key_type:* KeyType = *KeyType.K1*, *path: str = ''*, *compressed: bool = True*, *hexed: bool = True*) → str

Get the public key from the card.

Requires

- PIN code or challenge-response validated, except for PIN-less path
- Seed must exist

Parameters

- **derivation** (Derivation) – Derivation to use.
- **key_type** (KeyType) – Key type to use
- **path** (*str*)
- **compressed** (*bool*) – The returned value is in compressed format.

Returns

The public key for the given path in hexadecimal string format

Return type

str

Raises

- ***DerivationSelectionException*** – Card is not initialized with seed
- ***ReadPublicKeyException*** – Invalid data received from card

get_public_key_extended(*key_type:* KeyType = *KeyType.K1*, *puk: str = ''*) → str

Get the extended public key (xpub) from the card.

Requires

- PIN code or challenge-response validated
- Seed must exist
- XPUB capability must be enabled (or PUK provided to enable it)

Parameters

- **key_type** (KeyType) – Key type to use (default: KeyType.K1)
- **puk** (*str*) – Optional PUK code to enable XPUB capability

Returns

Extended public key in hexadecimal string format

Return type
str

Raises

- *exceptions.SeedException* – If no seed exists on the card
- *exceptions.ReadPublicKeyException* – If invalid data received from card

get_public_key_clear(*derivation: int*, *path: str = ''*, *compressed: bool = True*) → bytes

Get the public key within clear channel

Parameters

- **derivation** – Derivation + KeyType (e.g., Derivation.CURRENT_KEY + KeyType.K1)
- **path** – Optional BIP path string like "m/44'/0'/0'"
- **compressed** – Whether to return compressed format

Returns
Public key in bytes format

Raises
exceptions.ReadPublicKeyException – If bad data received

decrypt(*p1: int*, *pubkey: bytes*, *encrypted_data: bytes = b''*, *pin: str = ''*) → bytes

Decrypt data using ECIES (Elliptic Curve Integrated Encryption Scheme).

Generates a symmetric secret for simplified ECIES using an EC key in the BIP32 tree. This command is inspired by DECipher in OpenPGP smartcards. This allows asymmetric encryption using a key in the card seed tree. Anyone can encrypt with a public key from the card, and only the (private) key in the card can decrypt.

During the seed loading, the card saves in a dedicated key slot the result of a fixed derivation path. The child EC key used for this command is fixed (relative to a given seed). The path is computed with SLIP17 for the URI "openpgp://cryptnox" with index=0, and equals: m/17'/910196630'/2006168372'/332516148'/580566270'

The symmetric key is computed as SHA2(ECDH): SHA2-256(k.PubKey), where k is the private ECP256r1 key, the "decrypt" key slot.

Parameters

- **p1** (*int*) – 0x00 = Output symmetric key, 0x01 = Decrypt data in card
- **pubkey** (*bytes*) – Third party public key in X9.62 uncompressed format (0x04|X|Y, 65 bytes)
- **encrypted_data** (*bytes*) – Encrypted data (required when p1=1, must be 16-byte aligned)
- **pin** (*str*) – PIN code (required if no user key auth, right-padded with 0x00)

Returns
Symmetric key (p1=0) or decrypted data (p1=1)

Return type
bytes

Raises

- *exceptions.SeedException* – If no seed/key loaded
- *exceptions.PinException* – If PIN is incorrect
- *exceptions.DataValidationException* – If data length is incorrect
- *exceptions.GenericException* – If other errors occur

history(*index: int = 0*) → NamedTuple

Get history of hashes the card has signed regardless of any parameters given to sign

Requires

- PIN code or challenge-response validated

Parameters

index (*int*) – Index of entry in history

Returns

Return entry containing signing_counter, representing index of sign call, and hashed_data, the data that was signed

Return type

NamedTuple

property initialized: bool

Whether the card is initialized :rtype: bool

Type

return

load_wrapped_seed(*seed: bytes*, *pin: str = ''*) → None

load_seed(*seed: bytes*, *pin: str = ''*) → None

Load the given seed into the Cryptnox card.

Requires

- PIN code or challenge-response validated

Parameters

- **seed** (*bytes*) – Seed to initialize the card with
- **pin** (*str, optional*) – PIN code of the card. Can be empty if card is opened with challenge-response validation

Raises

KeyGenerationException – Data is not correct

property pin_authentication: bool

Whether the PIN code can be used for authentication :rtype: bool

Type

return

property pinless_enabled: bool

Return whether the card has a pinless path :rtype: bool

Type

return

reset(*puk: str*) → None

Reset the card and return it to factory settings.

Parameters
puk – PUK code associated with the card

property seed_source: *SeedSource*

How the seed was generated :rtype: SeedSource

Type
return

set_pin_authentication(*status: bool*, *puk: str*) → None

Turn on/off authentication with the PIN code. Other methods can still be used.

Parameters

- **status** (*bool*) – Status of PIN authentication
- **puk** (*str*) – PUK code associated with the card

Raises

- ***DataValidationException*** – input data is not valid
- ***PukException*** – PUK code is not valid

set_pinless_path(*path: str*, *puk: str*) → None

Define a BIP-32 derivation path whose key can sign without PIN entry.

This is a deliberate design feature for payment/NFC tap-to-pay use cases where user interaction is not possible. The operation is PUK-protected: only the card owner who knows the PUK can enable or change the pinless path, limiting the attack surface to that single derivation path.

Parameters

- **path** (*str*) – BIP-32 path to allow pinless signing (empty to clear)
- **puk** (*str*) – PUK code authorising the change

Raises

- ***SeedException*** – No seed loaded on the card
- ***PukException*** – PUK is invalid
- ***DataValidationException*** – Path format is invalid

set_extended_public_key(*status: bool*, *puk: str*) → None

Set extended public key capability.

This is a convenience wrapper around set_pubexport(status, 0, puk). Use set_pubexport() directly for more control.

property signing_counter: int

Counter of how many times the card has been used to sign :rtype: int

Type
return

user_key_add(*slot:* SlotIndex, *data_info: str*, *public_key: bytes*, *puk_code: str*, *cred_id: bytes = b''*) → None

Add user public key into the card for user authentication

Parameters

- **slot** (*int*) – Slot to write the public key to 1 - EC256R1 2 - RSA key, 2048 bits, public exponent must be 65537 3 - FIDO key
- **data_info** (*bytes*) – 64 bytes of user data
- **public_key** (*bytes*) – Public key of the secure element to be used for authentication
- **puk_code** (*str*) – PUK code of the card
- **cred_id** (*bytes, optional*) – Cred id. Used for FIDO2 authentication

Raises

DataValidationException – Invalid input data

user_key_delete(*slot:* SlotIndex, *puk_code: str*) → None

Delete the user key from slot and free up for insertion

Parameters

- **slot** (SlotIndex) – Slot to remove the key from
- **puk_code** (*str*) – PUK code of the card

Raises

DataValidationException – Invalid input data

user_key_info(*slot:* SlotIndex) → Tuple[str, str]

Get the description and public key of the user key

Requires

- PIN code or challenge-response validated

Parameters

slot (SlotIndex) – Index of slot for which to fetch the description

Returns

Description and public key in slot

Return type

tuple[str, str]

user_key_enabled(*slot_index:* SlotIndex)

Check if user key is present in given slot

Parameters

slot_index (SlotIndex) – Slot index to check for

Returns

Whether the user key for slot is present

Return type

bool

user_key_challenge_response_nonce() → bytes

Get 32 bytes random value from the card that is used to open the card with a user key

Take nonce value from the card. Sign it with a third party application, like TPM. Send the signature back into the card using *user_key_challenge_response_open()*

Returns
32 bytes random value used as nonce

Return type
bytes

user_key_challenge_response_open(*slot:* SlotIndex, *signature: bytes*) → bool

Send the nonce signature to the card to open it for operations, like it was opened by a PIN code

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **signature** (*bytes*) – Signature generated by a third party like TPM.

Returns
Whether the challenge response authentication succeeded

Return type
bool

Raises
DataValidationException – invalid input data

user_key_signature_open(*slot:* SlotIndex, *message: bytes*, *signature: bytes*) → bool

Used for opening the card to sign the given message

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **message** (*bytes*) – Message that will be sent to sign operation
- **signature** (*bytes*) – Signature generated by a third party, like TPM, on the same message

Returns
Whether the challenge response authentication succeeded

Return type
bool

Raises
DataValidationException – invalid input data

generate_seed_wrapper(*size: int = 2048*) → bytes

sign_public(*key_type:* KeyType = *KeyType.K1*) → bytes

sign(*data: bytes*, *derivation:* Derivation = *Derivation.CURRENT_KEY*, *key_type:* KeyType = *KeyType.K1*, *path: str = ''*, *pin: str = ''*, *filter_eos: bool = False*) → bytes

Sign the message using given derivation.

Requires

- PIN code provided, authenticate with user key by signing same message or PIN-less path used
- Seed must be loaded

Parameters

- **data** (*bytes*) – Data to sign
- **derivation** (Derivation) – Derivation to use.
- **key_type** (KeyType, *optional*) – Key type to use. Defaults to K1
- **path** (*str, optional*) – Path of the key. If empty use main key
- **pin** (*str, optional*) – PIN code of the card
- **filter_eos** (*str, optional*) – Filter signature so it is valid for EOS network, might take longer. Defaults to False

Returns
The signature generated by the card in DER common format.

Return type
bytes

Raises
DataException – Invalid data received during signature

property valid_key: bool
Check if the card has a valid key

Returns
Whether the card has a valid key.

Return type
bool

static valid_puk(*puk: str*, *puk_name: str = 'puk'*) → str
Check if provided puk is valid

Parameters

- **puk** (*str*) – The puk to check if valid (accepts all ASCII characters)
- **puk_name** (*str, optional*) – Value used in DataValidationException for puk name. Defaults to: puk

Return str
Provided puk in str format if valid

Raises
DataValidationException – Provided puk is not valid (wrong length or non-ASCII characters)

signature_check(*nonce: bytes*) → *SignatureCheckResult*
Sign random 32 bytes for validation that private key of public key is on the card.

BasicG1 cards do not support this operation. Use NFT card instead.

Raises
NotImplementedError – BasicG1 cards do not support signature_check

verify_pin(*pin: str | None = None*) → int | None

Verify PIN code, or query remaining retries without decrementing the counter.

If `pin` is `None`, returns the number of PIN attempts remaining (safe read). If `pin` is provided, verifies it and opens the card for protected operations.

Parameters

pin (*str or None*) – PIN code, or `None` to query remaining retries.

Returns

Retries remaining when `pin` is `None`, otherwise `None`.

Return type

int or None

Raises

- ***PinException*** – Invalid PIN code
- ***DataValidationException*** – Invalid length or PIN code authentication disabled
- ***SoftLock*** – The card has been locked and needs power cycling before it can be used again

set_pubexport(*status: bool*, *p1: int*, *puk: str*) → None

Set pubexport capability (xpub or clear pubkey)

Parameters

- **status** – True to enable, False to disable
- **p1** – 0 for xpub, 1 for clear pubkey
- **puk** – PUK code

Raises

- ***exceptions.DataValidationException*** – If p1 is invalid
- ***exceptions.PukException*** – If PUK is incorrect

set_xpubread(*status: bool*, *puk: str*) → None

Set extended public key read capability

Parameters

- **status** – True to enable, False to disable
- **puk** – PUK code

set_clearpubkey(*status: bool*, *puk: str*) → None

Set clear public key read capability

Parameters

- **status** – True to enable, False to disable
- **puk** – PUK code

class cryptnox_sdk_py.card.**Nft**(**args*, ***kwargs*)

Bases: *BasicG1*

Class containing functionality for NFT card which has limited capabilities

type = 78

__init__(**args*, ***kwargs*)

derive(*key_type:* KeyType = *KeyType.K1*, *path: str* = *''*)
Derive key on path and make it the current key in the card

Requires

- PIN code or challenge-response validated
- Seed must exist

Parameters

- **key_type** (KeyType) – Key type to do derive on
- **path** (*str*) – Path on which to do derivation

get_public_key(*derivation:* Derivation = *Derivation.CURRENT_KEY*, *key_type:* KeyType = *KeyType.K1*, *path: str* = *''*, *compressed: bool* = *False*, *hexed: bool* = *True*) → str
Get the public key from the card.

Requires

- PIN code or challenge-response validated, except for PIN-less path
- Seed must exist

Parameters

- **derivation** (Derivation) – Derivation to use.
- **key_type** (KeyType) – Key type to use
- **path** (*str*)
- **compressed** (*bool*) – The returned value is in compressed format.

Returns
The public key for the given path in hexadecimal string format

Return type
str

Raises

- ***DerivationSelectionException*** – Card is not initialized with seed
- ***ReadPublicKeyException*** – Invalid data received from card

get_public_key_clear(*derivation: int*, *path: str* = *''*, *compressed: bool* = *True*) → bytes
Get the public key within clear channel

Parameters

- **derivation** – Derivation + KeyType (e.g., Derivation.CURRENT_KEY + KeyType.K1)
- **path** – Optional BIP path string like “m/44’/0’/0’”
- **compressed** – Whether to return compressed format

Returns
Public key in bytes format

Raises
exceptions.ReadPublicKeyException – If bad data received

set_pubexport(*status: bool*, *p1: int*, *puk: str*) → None
Set pubexport capability (xpub or clear pubkey)

Parameters

- **status** – True to enable, False to disable
- **p1** – 0 for xpub, 1 for clear pubkey
- **puk** – PUK code

Raises

- *exceptions.DataValidationException* – If p1 is invalid
- *exceptions.PukException* – If PUK is incorrect

generate_random_number(*size: int*) → bytes
Generate random number on the car and return it.

Parameters
size (*int*) – Output data size in bytes (between 16 and 64, mod 4)

Returns
Random number generated by the chip

Return type
bytes

Raises
DataValidationException – size in not a number between 16 and 64 or is not divisible by 4

load_seed(*seed: bytes*, *pin: str = ''*) → None
Load the given seed into the Cryptnox card.

Requires

- PIN code or challenge-response validated

Parameters

- **seed** (*bytes*) – Seed to initialize the card with
- **pin** (*str, optional*) – PIN code of the card. Can be empty if card is opened with challenge-response validation

Raises
KeyGenerationException – Data is not correct

set_pin_authentication(*status: bool*, *puk: str*) → None
Turn on/off authentication with the PIN code. Other methods can still be used.

Parameters

- **status** (*bool*) – Status of PIN authentication

- **puk** (*str*) – PUK code associated with the card

Raises

- ***DataValidationException*** – input data is not valid
- ***PukException*** – PUK code is not valid

set_pinless_path(*path: str*, *puk: str*) → None

Define a BIP-32 derivation path whose key can sign without PIN entry.

This is a deliberate design feature for payment/NFC tap-to-pay use cases where user interaction is not possible. The operation is PUK-protected: only the card owner who knows the PUK can enable or change the pinless path, limiting the attack surface to that single derivation path.

Parameters

- **path** (*str*) – BIP-32 path to allow pinless signing (empty to clear)
- **puk** (*str*) – PUK code authorising the change

Raises

- ***SeedException*** – No seed loaded on the card
- ***PukException*** – PUK is invalid
- ***DataValidationException*** – Path format is invalid

user_key_add(*slot:* SlotIndex, *data_info: str*, *public_key: bytes*, *puk_code: str*, *cred_id: bytes = b''*) → None

Add user public key into the card for user authentication

Parameters

- **slot** (*int*) – Slot to write the public key to 1 - EC256R1 2 - RSA key, 2048 bits, public exponent must be 65537 3 - FIDO key
- **data_info** (*bytes*) – 64 bytes of user data
- **public_key** (*bytes*) – Public key of the secure element to be used for authentication
- **puk_code** (*str*) – PUK code of the card
- **cred_id** (*bytes, optional*) – Cred id. Used for FIDO2 authentication

Raises

DataValidationException – Invalid input data

user_key_delete(*slot:* SlotIndex, *puk_code: str*) → None

Delete the user key from slot and free up for insertion

Parameters

- **slot** (SlotIndex) – Slot to remove the key from
- **puk_code** (*str*) – PUK code of the card

Raises

DataValidationException – Invalid input data

user_key_info(*slot:* SlotIndex) → Tuple[str, str]

Get the description and public key of the user key

Requires

- PIN code or challenge-response validated

Parameters

slot (SlotIndex) – Index of slot for which to fetch the description

Returns

Description and public key in slot

Return type

tuple[str, str]

user_key_enabled(*slot_index:* SlotIndex)

Check if user key is present in given slot

Parameters

slot_index (SlotIndex) – Slot index to check for

Returns

Whether the user key for slot is present

Return type

bool

user_key_challenge_response_nonce() → bytes

Get 32 bytes random value from the card that is used to open the card with a user key

Take nonce value from the card. Sign it with a third party application, like TPM. Send the signature back into the card using *user_key_challenge_response_open()*

Returns

32 bytes random value used as nonce

Return type

bytes

user_key_challenge_response_open(*slot:* SlotIndex, *signature: bytes*) → bool

Send the nonce signature to the card to open it for operations, like it was opened by a PIN code

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **signature** (*bytes*) – Signature generated by a third party like TPM.

Returns

Whether the challenge response authentication succeeded

Return type

bool

Raises

DataValidationException – invalid input data

user_key_signature_open(*slot:* SlotIndex, *message: bytes*, *signature: bytes*) → bool

Used for opening the card to sign the given message

Parameters

- **slot** (SlotIndex) – Slot to use to open the card
- **message** (*bytes*) – Message that will be sent to sign operation
- **signature** (*bytes*) – Signature generated by a third party, like TPM, on the same message

Returns

Whether the challenge response authentication succeeded

Return type

bool

Raises

DataValidationException – invalid input data

signature_check(*nonce: bytes*) → *SignatureCheckResult*

Sign random 32 bytes for validation that private key of public key is on the card.

BasicG1 cards do not support this operation. Use NFT card instead.

Raises

NotImplementedError – BasicG1 cards do not support signature_check

The `cryptnox_sdk_py.card` package contains classes for various Cryptnox card types and exports:

class cryptnox_sdk_py.card.**Base**

Base card interface class. See *cryptnox_sdk_py.card.base.Base* for details.

class cryptnox_sdk_py.card.**BasicG1**

Basic Generation 1 card implementation. See *cryptnox_sdk_py.card.basic_g1.BasicG1* for details.

class cryptnox_sdk_py.card.**Nft**

NFT card implementation. See *cryptnox_sdk_py.card.nft.Nft* for details.

cryptnox_sdk_py.cryptos package

Submodules

cryptnox_sdk_py.cryptos.main module

Module containing core cryptographic functions for Cryptnox cards.

Provides elliptic curve cryptography (secp256k1), key generation, digital signatures, and cryptocurrency address generation.

cryptnox_sdk_py.cryptos.main.**change_curve**(*p*, *n*, *a*, *b*, *gx*, *gy*)

cryptnox_sdk_py.cryptos.main.**getG**()

cryptnox_sdk_py.cryptos.main.**inv**(*a*, *n*)

cryptnox_sdk_py.cryptos.main.**access**(*obj*, *prop*)

cryptnox_sdk_py.cryptos.main.**multiaccess**(*obj*, *prop*)

cryptnox_sdk_py.cryptos.main.**obj_slice**(*obj*, *start=0*, *end=2**200*)

cryptnox_sdk_py.cryptos.main.**count**(*obj*)

cryptnox_sdk_py.cryptos.main.**obj_sum**(*obj*)

cryptnox_sdk_py.cryptos.main.**isinf**(*p*)

cryptnox_sdk_py.cryptos.main.**to_jacobian**(*p*)

cryptnox_sdk_py.cryptos.main.**jacobian_double**(*p*)

cryptnox_sdk_py.cryptos.main.**jacobian_add**(*p*, *q*)

cryptnox_sdk_py.cryptos.main.**fast_add**(*a*, *b*)

cryptnox_sdk_py.cryptos.main.**get_pubkey_format**(*pub*)

cryptnox_sdk_py.cryptos.main.**encode_pubkey**(*pub*, *formt*)

cryptnox_sdk_py.cryptos.main.**decode_pubkey**(*pub*, *formt=None*)

cryptnox_sdk_py.cryptos.main.**get_privkey_format**(*priv*)

cryptnox_sdk_py.cryptos.main.**encode_privkey**(*priv*, *formt*, *vbyte=128*)

cryptnox_sdk_py.cryptos.main.**decode_privkey**(*priv*, *formt=None*)

cryptnox_sdk_py.cryptos.main.**add_pubkeys**(*p1*, *p2*)

cryptnox_sdk_py.cryptos.main.**add_privkeys**(*p1*, *p2*)

cryptnox_sdk_py.cryptos.main.**divide**(*pubkey*, *privkey*)

cryptnox_sdk_py.cryptos.main.**compress**(*pubkey*)

cryptnox_sdk_py.cryptos.main.**decompress**(*pubkey*)

cryptnox_sdk_py.cryptos.main.**neg_pubkey**(*pubkey*)

cryptnox_sdk_py.cryptos.main.**neg_privkey**(*privkey*)

cryptnox_sdk_py.cryptos.main.**subtract_pubkeys**(*p1*, *p2*)

cryptnox_sdk_py.cryptos.main.**subtract_privkeys**(*p1*, *p2*)

cryptnox_sdk_py.cryptos.main.**bin_hash160**(*string*)

cryptnox_sdk_py.cryptos.main.**hash160**(*string*)

cryptnox_sdk_py.cryptos.main.**hex_to_hash160**(*s_hex*)

cryptnox_sdk_py.cryptos.main.**bin_sha256**(*string*)

cryptnox_sdk_py.cryptos.main.**sha256**(*string*)

cryptnox_sdk_py.cryptos.main.**bin_ripemd160**(*string*)

cryptnox_sdk_py.cryptos.main.**ripemd160**(*string*)

cryptnox_sdk_py.cryptos.main.**bin_dbl_sha256**(*s*)

cryptnox_sdk_py.cryptos.main.**dbl_sha256**(*string*)

cryptnox_sdk_py.cryptos.main.**bin_slowsha**(*string*)

cryptnox_sdk_py.cryptos.main.**slowsha**(*string*)

cryptnox_sdk_py.cryptos.main.**hash_to_int**(*x*)

cryptnox_sdk_py.cryptos.main.**num_to_var_int**(*x*)

cryptnox_sdk_py.cryptos.main.**electrum_sig_hash**(*message*)

cryptnox_sdk_py.cryptos.main.**b58check_to_bin**(*inp*)

cryptnox_sdk_py.cryptos.main.**get_version_byte**(*inp*)

cryptnox_sdk_py.cryptos.main.**hex_to_b58check**(*inp*, *magicbyte=0*)

cryptnox_sdk_py.cryptos.main.**b58check_to_hex**(*inp*)

cryptnox_sdk_py.cryptos.main.**pubkey_to_hash**(*pubkey*)

cryptnox_sdk_py.cryptos.main.**pubkey_to_hash_hex**(*pubkey*)

cryptnox_sdk_py.cryptos.main.**pubkey_to_address**(*pubkey*, *magicbyte=0*)

cryptnox_sdk_py.cryptos.main.**pubtoaddr**(*pubkey*, *magicbyte=0*)

cryptnox_sdk_py.cryptos.main.**is_privkey**(*priv*)

cryptnox_sdk_py.cryptos.main.**is_pubkey**(*pubkey*)

cryptnox_sdk_py.cryptos.main.**encode_sig**(*v*, *r*, *s*)

cryptnox_sdk_py.cryptos.main.**decode_sig**(*sig*)

cryptnox_sdk_py.cryptos.main.**deterministic_generate_k**(*msghash*, *priv*)

cryptnox_sdk_py.cryptos.main.**ecdsa_verify_addr**(*msg*, *sig*, *addr*, *coin*)

cryptnox_sdk_py.cryptos.main.**ecdsa_verify**(*msg*, *sig*, *pub*, *coin*)

cryptnox_sdk_py.cryptos.main.**add**(*p1*, *p2*)

cryptnox_sdk_py.cryptos.main.**subtract**(*p1*, *p2*)

cryptnox_sdk_py.cryptos.main.**magicbyte_to_prefix**(*magicbyte*)

cryptnox_sdk_py.cryptos.py2specials module

Module containing Python 2 compatibility utilities.

Provides type definitions, string/bytes handling, and encoding utilities for Python 2 compatibility in cryptographic operations.

cryptnox_sdk_py.cryptos.py2specials.**bin_dbl_sha256**(*s*)

cryptnox_sdk_py.cryptos.py2specials.**get_code_string**(*base*)

cryptnox_sdk_py.cryptos.py2specials.**lpad**(*msg*, *symbol*, *length*)

cryptnox_sdk_py.cryptos.py2specials.**encode**(*val*, *base*, *minlen=0*)

cryptnox_sdk_py.cryptos.py2specials.**decode**(*string*, *base*)

cryptnox_sdk_py.cryptos.py2specials.**changebase**(*string*, *frm*, *to*, *minlen=0*)

cryptnox_sdk_py.cryptos.py2specials.**bin_to_b58check**(*inp*, *magicbyte=0*)

cryptnox_sdk_py.cryptos.py2specials.**bytes_to_hex_string**(*b*)

cryptnox_sdk_py.cryptos.py2specials.**safe_from_hex**(*s*)

cryptnox_sdk_py.cryptos.py2specials.**from_int_representation_to_bytes**(*a*)

cryptnox_sdk_py.cryptos.py2specials.**from_int_to_byte**(*a*)

cryptnox_sdk_py.cryptos.py2specials.**from_byte_to_int**(*a*)

cryptnox_sdk_py.cryptos.py2specials.**from_bytes_to_string**(*s*)

cryptnox_sdk_py.cryptos.py2specials.**from_string_to_bytes**(*a*)

cryptnox_sdk_py.cryptos.py2specials.**safe_hexlify**(*a*)

cryptnox_sdk_py.cryptos.py2specials.**random_string**(*x*)

cryptnox_sdk_py.cryptos.py2specials.**from_jacobian**(*p*)
Convert Jacobian coordinates to affine coordinates.

cryptnox_sdk_py.cryptos.py3specials module

Module containing Python 3 specific cryptographic implementations.

Provides Python 3 optimized cryptographic functions including elliptic curve operations, ECDSA signing/verification, and mathematical utilities for secp256k1 operations.

cryptnox_sdk_py.cryptos.py3specials.**bin_dbl_sha256**(*s*)

cryptnox_sdk_py.cryptos.py3specials.**lpad**(*msg*, *symbol*, *length*)

cryptnox_sdk_py.cryptos.py3specials.**get_code_string**(*base*)

cryptnox_sdk_py.cryptos.py3specials.**changebase**(*string*, *frm*, *to*, *minlen=0*)

cryptnox_sdk_py.cryptos.py3specials.**bin_to_b58check**(*inp*, *magicbyte=0*)

cryptnox_sdk_py.cryptos.py3specials.**bytes_to_hex_string**(*b*)

cryptnox_sdk_py.cryptos.py3specials.**safe_from_hex**(*s*)

cryptnox_sdk_py.cryptos.py3specials.**from_int_representation_to_bytes**(*a*)

cryptnox_sdk_py.cryptos.py3specials.**from_int_to_byte**(*a*)

cryptnox_sdk_py.cryptos.py3specials.**from_byte_to_int**(*a*)

cryptnox_sdk_py.cryptos.py3specials.**from_string_to_bytes**(*a*)

cryptnox_sdk_py.cryptos.py3specials.**safe_hexlify**(*a*)

cryptnox_sdk_py.cryptos.py3specials.**encode**(*val*, *base*, *minlen=0*)

cryptnox_sdk_py.cryptos.py3specials.**decode**(*string*, *base*)

cryptnox_sdk_py.cryptos.py3specials.**random_string**(*x*)

cryptnox_sdk_py.cryptos.py3specials.**from_jacobian**(*p*)
Convert Jacobian coordinates to affine coordinates.

cryptnox_sdk_py.cryptos.py3specials.**multiply**(*pubkey*, *privkey*)
Multiply a public key by a scalar (private key).

cryptnox_sdk_py.cryptos.py3specials.**ecdsa_raw_verify**(*msghash*, *sig*, *pub*)
Verify ECDSA signature.

cryptnox_sdk_py.cryptos.py3specials.**ecdsa_recover**(*msg*, *sig*)
Recover public key from ECDSA signature.

cryptnox_sdk_py.cryptos.ripemd module

Module containing RIPEMD-160 hash algorithm implementation.

Pure Python implementation of the RIPEMD-160 cryptographic hash function. Used for generating Bitcoin addresses and other cryptographic operations.

class cryptnox_sdk_py.cryptos.ripemd.**RIPEMD160**(*arg=None*)
Bases: `object`

Return a new RIPEMD160 object. An optional string argument may be provided; if present, this string will be automatically hashed.

__init__(*arg=None*)

update(*arg*)

digest()

hexdigest()

copy()

cryptnox_sdk_py.cryptos.ripemd.**new**(*arg=None*)
Return a new RIPEMD160 object. An optional string argument may be provided; if present, this string will be automatically hashed.

class cryptnox_sdk_py.cryptos.ripemd.**RMDContext**

Bases: object

__init__()

copy()

cryptnox_sdk_py.cryptos.ripemd.**ROL**(*n*, *x*)

cryptnox_sdk_py.cryptos.ripemd.**F0**(*x*, *y*, *z*)

cryptnox_sdk_py.cryptos.ripemd.**F1**(*x*, *y*, *z*)

cryptnox_sdk_py.cryptos.ripemd.**F2**(*x*, *y*, *z*)

cryptnox_sdk_py.cryptos.ripemd.**F3**(*x*, *y*, *z*)

cryptnox_sdk_py.cryptos.ripemd.**F4**(*x*, *y*, *z*)

cryptnox_sdk_py.cryptos.ripemd.**R**(*a*, *b*, *c*, *d*, *e*, *Fj*, *Kj*, *sj*, *rj*, *X*)

cryptnox_sdk_py.cryptos.ripemd.**RMD160Transform**(*state*, *block*)

cryptnox_sdk_py.cryptos.ripemd.**RMD160Update**(*ctx*, *inp*, *inplen*)

cryptnox_sdk_py.cryptos.ripemd.**RMD160Final**(*ctx*)

Module contents

Module containing cryptographic utilities for Cryptnox cards.

cryptnox_sdk_py.cryptos.**encode_pubkey**(*pub*, *formt*)

The cryptnox_sdk_py.cryptos package contains cryptographic utilities for Cryptnox cards and exports:

Main cryptographic operations module. See *cryptnox_sdk_py.cryptos.main* for details.

Python 2 specific cryptographic utilities. See *cryptnox_sdk_py.cryptos.py2specials* for details.

Python 3 specific cryptographic utilities. See *cryptnox_sdk_py.cryptos.py3specials* for details.

py3specials.**encode_pubkey**(*pubkey*, *formt*)

Encode a public key in the specified format. See *cryptnox_sdk_py.cryptos.main.encode_pubkey()* for details.

2.1.2 Submodules

2.1.3 cryptnox_sdk_py.binary_utils module

Utility module for handling binary data

cryptnox_sdk_py.binary_utils.**list_to_hexadecimal**(*data: List[int]*, *sep: str = ''*) → str

Convert list of integers into hexadecimal representation

Parameters

- **data** (*List[int]*) – List of integer to return in hexadecimal string form
- **sep** (*str*) – (optional) Separator to use to join the hexadecimal numbers

Returns
list

Return type
str

cryptnox_sdk_py.binary_utils.**hexadecimal_to_list**(*value: str*) → List[int]

Convert given string containing hexadecimal representation of numbers into list of integers

Parameters
value (*string*) – String containing hexadecimal numbers

Returns
List of hexadecimal values in integer form

Return type
List[int]

cryptnox_sdk_py.binary_utils.**path_to_bytes**(*path_str: str*) → bytes

Convert given path for format that the card uses

Parameters
path_str (*str*) – path to convert

Returns
path formatted for use with the card.

Return type
bytes

cryptnox_sdk_py.binary_utils.**binary_to_list**(*data: bytes*) → List[int]

Convert given binary data to it's representation as a list of hexadecimal values.

Parameters
data (*bytes*) – Binary data to convert to

Returns
List containing data in hexadecimal numbers in integer format

Return type
List[int]

cryptnox_sdk_py.binary_utils.**pad_data**(*data: bytes*) → bytes

Pad data with 0s to be length of 128.

Parameters
data – Data to be padded.

Returns
Data padded with 0s with length 128.

Return type
bytes

cryptnox_sdk_py.binary_utils.**remove_padding**(*data: bytes*) → bytes

Remove padding from the data

Parameters
data (*bytes*) – Data from which to remove padding

Returns
Data without the padding

Return type
bytes

2.1.4 cryptnox_sdk_py.connection module

Module for keeping the connection to the reader.

Sending and receiving information from the card through the reader.

class cryptnox_sdk_py.connection.**Connection**(*index: int = 0*, *debug: bool = False*, *conn: List = None*, *remote: bool = False*)

Bases: ContextDecorator

Connection to the reader.

Sends and receives messages from the card using the reader.

Parameters

- **index** (*int*) – Index of the reader to initialize the connection with
- **debug** (*bool*) – Show debug information during requests
- **conn** (*List*) – List of sockets to use for remote connections
- **remote** (*bool*) – Use remote sockets for communications with the cards

Variables
self.card (Card) – Information about the card.

__init__(*index: int = 0*, *debug: bool = False*, *conn: List = None*, *remote: bool = False*)

disconnect() → None
Disconnect from the card reader and clean up the connection.

This method properly closes the connection to the card reader without deleting the Connection object itself.

send_apdu(*apdu: List[int]*) → Tuple[List[int], int, int]
Send data to the card in plain format

Parameters
apdu (*int*) – list of the APDU header

Return bytes
Result of the query that was sent to the card

Return type
bytes

Raises
ConnectionException – Issue in the connection

send_encrypted(*apdu: List[int]*, *data: bytes*, *receive_long: bool = False*) → bytes

Send data to the card in encrypted format

Parameters

- **apdu** (*int*) – list of the APDU header
- **data** – bytes of the data payload (in clear, will be encrypted)
- **receive_long** (*bool*)

Return bytes
Result of the query that was sent to the card

Return type
bytes

Raises
CryptnoxException – General exceptions

remote_read(*apdu: List[int]*) → Tuple[List[int], int, int]

2.1.5 cryptnox_sdk_py.crypto_utils module

Module containing cryptographic methods used by the library

cryptnox_sdk_py.crypto_utils.**aes_encrypt**(*key: bytes*, *initialization_vector: bytes*, *data: bytes*) → bytes

AES encrypt data using key and initialization vector.

Parameters

- **key** (*bytes*) – Key to use for encryption
- **initialization_vector** (*bytes*) – Initialization vector
- **data** (*bytes*) – Data to encrypt

Returns
Encrypted data

Return type
bytes

cryptnox_sdk_py.crypto_utils.**aes_decrypt**(*key: bytes*, *initialization_vector: bytes*, *data: bytes*) → bytes

AES decrypt data using key and initialization vector.

Parameters

- **key** (*bytes*) – Key to use for encryption
- **initialization_vector** (*bytes*) – Initialization vector
- **data** (*bytes*) – Data to decrypt

Returns
Decrypted data

Return type
bytes

2.1.6 cryptnox_sdk_py.enums module

Enum classes used by the module

class cryptnox_sdk_py.enums.**AuthType**(**values*)
Bases: Enum

Predefined values for authentication type.

NO_AUTH = 0

PIN = 1

USER_KEY = 2

class cryptnox_sdk_py.enums.**Derivation**(**values*)
Bases: IntEnum

Predefined values to use for parameters as Derivation.

CURRENT_KEY = 0

DERIVE = 1

DERIVE_AND_MAKE_CURRENT = 2

PINLESS_PATH = 3

class cryptnox_sdk_py.enums.**KeyType**(**values*)
Bases: IntEnum

Predefined values to use for parameters as KeyType.

K1 = 0

R1 = 16

class cryptnox_sdk_py.enums.**Origin**(**values*)
Bases: Enum

Predefined values for keeping the origin of the card

UNKNOWN = 0

ORIGINAL = 1

FAKE = 2

class cryptnox_sdk_py.enums.**SlotIndex**(**values*)
Bases: IntEnum

Predefined values to use for parameters as SlotIndex.

EC256R1 = 1

RSA = 2

FIDO = 3

class cryptnox_sdk_py.enums.**SeedSource**(**values*)
Bases: Enum

Predefined values for how seed was created

NO_SEED = 0

SINGLE = 75

EXTENDED = 88

EXTERNAL = 76

INTERNAL = 83

DUAL = 68

WRAPPED = 82

2.1.7 cryptnox_sdk_py.exceptions module

Module containing all exceptions that Cryptnox SDK Python module can raise.

exception cryptnox_sdk_py.exceptions.**CryptnoxException**
Bases: Exception

Base exception for the class exceptions.

exception cryptnox_sdk_py.exceptions.**CardClosedException**
Bases: Exception

The card wasn't opened with PIN code or challenge-response

exception cryptnox_sdk_py.exceptions.**CardException**
Bases: *CryptnoxException*

No card was detected in the card reader.

exception cryptnox_sdk_py.exceptions.**CardTypeException**
Bases: *CryptnoxException*

The detected card is not supported by this library

exception cryptnox_sdk_py.exceptions.**CertificateException**
Bases: *CryptnoxException*

There was an issue with the certification

exception cryptnox_sdk_py.exceptions.**ConnectionException**
Bases: *CryptnoxException*

An issue occurred in the communication with the reader

exception cryptnox_sdk_py.exceptions.**DataException**
Bases: *CryptnoxException*

The reader returned an empty message.

exception cryptnox_sdk_py.exceptions.**DataValidationException**
Bases: *CryptnoxException*

The sent data is not valid.

exception cryptnox_sdk_py.exceptions.**DerivationSelectionException**
Bases: *CryptnoxException*

Not a valid derivation selection.

exception cryptnox_sdk_py.exceptions.**KeySelectionException**
Bases: *CryptnoxException*

Not a valid key type selection

exception cryptnox_sdk_py.exceptions.**EOSKeyError**
Bases: *CryptnoxException*

The signature wasn't compatible with EOS standard after 10 tries

exception cryptnox_sdk_py.exceptions.**FirmwareException**
Bases: *CryptnoxException*

There is an issue with the firmware on the card

exception cryptnox_sdk_py.exceptions.**GenuineCheckException**
Bases: *CryptnoxException*

The detected card is not a genuine Cryptnox product.

exception cryptnox_sdk_py.exceptions.**GenericException**(*status: bytes*)
Bases: *CryptnoxException*

Generic exception that can mean multiple things depending on the call to the card

Process stats and throw a specific Exception from it.

__init__(*status: bytes*)

exception cryptnox_sdk_py.exceptions.**InitializationException**
Bases: *CryptnoxException*

The card hasn't been initialized.

exception cryptnox_sdk_py.exceptions.**KeyAlreadyGenerated**
Bases: *CryptnoxException*

Key can not be generated twice.

exception cryptnox_sdk_py.exceptions.**SeedException**
Bases: *CryptnoxException*

Keys weren't found on the card.

exception cryptnox_sdk_py.exceptions.**KeyGenerationException**
Bases: *CryptnoxException*

Error in key generation.

exception cryptnox_sdk_py.exceptions.**PinAuthenticationException**

Bases: *CryptnoxException*

Error in turning off PIN authentication. There is no user key in the card

exception cryptnox_sdk_py.exceptions.**PinBlockedException**

Bases: *CryptnoxException*

PIN is locked. Use the unlock_pin command to unlock it before attempting this operation.

exception cryptnox_sdk_py.exceptions.**PinException**(*message: str = 'Invalid PIN code was provided'*, *number_of_retries: int = 0*)

Bases: *CryptnoxException*

Sent PIN code is not valid.

Parameters

- **number_of_retries** (*int*) – Number of retries to send the PIN code before the card is locked.
- **message** (*str*) – Optional message

__init__(*message: str = 'Invalid PIN code was provided'*, *number_of_retries: int = 0*)

exception cryptnox_sdk_py.exceptions.**PukException**(*message: str = 'Invalid PUK code was provided'*, *number_of_retries: int = 0*)

Bases: *CryptnoxException*

Sent PUK code is not valid.

Parameters

- **number_of_retries** (*int*) – Number of retries to send the PIN code before the card is locked.
- **message** (*str*) – Optional message

__init__(*message: str = 'Invalid PUK code was provided'*, *number_of_retries: int = 0*)

exception cryptnox_sdk_py.exceptions.**ReadPublicKeyException**

Bases: *CryptnoxException*

Data received during public key reading is not valid.

exception cryptnox_sdk_py.exceptions.**ReaderException**

Bases: *CryptnoxException*

Card reader wasn't found attached to the device.

exception cryptnox_sdk_py.exceptions.**SecureChannelException**

Bases: *CryptnoxException*

Secure channel couldn't be established.

exception cryptnox_sdk_py.exceptions.**SoftLock**

Bases: *CryptnoxException*

The card is soft locked, and requires power cycle before it can be opened

exception cryptnox_sdk_py.exceptions.**CardNotBlocked**

Bases: *CryptnoxException*

Trying to unlock unblocked card

2.1.8 cryptnox_sdk_py.factory module

Module for getting Cryptnox cards information and getting instance of card from connection

cryptnox_sdk_py.factory.**get_card**(*connection:* Connection, *debug: bool = False*) → *Base*

Get card instance that is using given connection.

Parameters

- **connection** (Connection) – Connection to use for operation
- **debug** (*bool*) – Prints information about communication

Returns
Instance of card

Return type
Base

Raises
CardException – Card with given serial number not found

2.1.9 cryptnox_sdk_py.reader module

Module that handles different card reader types and their drivers.

exception cryptnox_sdk_py.reader.**ReaderException**

Bases: Exception

Reader hasn't been found or other reader related issues

exception cryptnox_sdk_py.reader.**CardException**

Bases: Exception

The reader is present but there is an issue in connecting to the card

exception cryptnox_sdk_py.reader.**ConnectionException**

Bases: Exception

An issue has occurred in the communication with the card.

class cryptnox_sdk_py.reader.**Reader**

Bases: object

Abstract class describing methods to be implemented. Holds the connection.

__init__()

abstractmethod connect() → None

Connect to the card found in the selected reader.

Returns
None

abstractmethod send(*apdu: List[int]*) → Tuple[List[str], int, int]

Send APDU to the reader and card and retrieve the result with status codes.

Parameters
apdu (*List[int]*) – Command to be sent

Returns
Return the result of the query and two status codes

Return type
Tuple[List[str], int, int]

bool() → bool

Is there an active connection

Return type
Is there an active connection

Returns
bool

disconnect() → None

Disconnect from the card.

Returns
None

class cryptnox_sdk_py.reader.**NfcReader**

Bases: *Reader*

Specialized reader using xantares/nfc-binding

__init__()

connect()

Connect to the card found in the selected reader.

Returns
None

send(*apdu: List[int]*) → Tuple[List[str], int, int]

Send APDU to the reader and card and retrieve the result with status codes.

Parameters
apdu (*List[int]*) – Command to be sent

Returns
Return the result of the query and two status codes

Return type
Tuple[List[str], int, int]

class cryptnox_sdk_py.reader.**SmartCard**(*index: int = 0*)

Bases: *Reader*

Generic smart card reader class

Parameters
index (*int*) – Index of the reader to initialize.

__init__(*index: int = 0*)

connect() → None
Connect to the card found in the selected reader.

Returns
None

send(*apdu: List[int]*) → Tuple[List[str], int, int]
Send APDU to the reader and card and retrieve the result with status codes.

Parameters
apdu (*List[int]*) – Command to be sent

Returns
Return the result of the query and two status codes

Return type
Tuple[List[str], int, int]

disconnect() → None
Disconnect from the card.

Returns
None

cryptnox_sdk_py.reader.**get**(*index: int = 0*) → *Reader*
Get the reader that can be found on the given position.

Parameters
index (*int*) – Index of reader to be initialized and used

Returns
Reader object that can be used.

Return type
Reader

2.1.10 Module contents

This is a library for communicating with Cryptnox cards

See the README.md for API details and general information.

cryptnox_sdk_py.**Card**
alias of *Base*

class cryptnox_sdk_py.**Connection**(*index: int = 0*, *debug: bool = False*, *conn: List = None*, *remote: bool = False*)

Bases: ContextDecorator

Connection to the reader.

Sends and receives messages from the card using the reader.

Parameters

- **index** (*int*) – Index of the reader to initialize the connection with
- **debug** (*bool*) – Show debug information during requests

- **conn** (*List*) – List of sockets to use for remote connections
- **remote** (*bool*) – Use remote sockets for communications with the cards

Variables
self.card (Card) – Information about the card.

__init__(*index: int = 0*, *debug: bool = False*, *conn: List = None*, *remote: bool = False*)

disconnect() → None
Disconnect from the card reader and clean up the connection.

This method properly closes the connection to the card reader without deleting the Connection object itself.

send_apdu(*apdu: List[int]*) → Tuple[List[int], int, int]
Send data to the card in plain format

Parameters
apdu (*int*) – list of the APDU header

Return bytes
Result of the query that was sent to the card

Return type
bytes

Raises
ConnectionException – Issue in the connection

send_encrypted(*apdu: List[int]*, *data: bytes*, *receive_long: bool = False*) → bytes
Send data to the card in encrypted format

Parameters

- **apdu** (*int*) – list of the APDU header
- **data** – bytes of the data payload (in clear, will be encrypted)
- **receive_long** (*bool*)

Return bytes
Result of the query that was sent to the card

Return type
bytes

Raises
CryptnoxException – General exceptions

remote_read(*apdu: List[int]*) → Tuple[List[int], int, int]

class cryptnox_sdk_py.**SlotIndex**(**values*)
Bases: IntEnum

Predefined values to use for parameters as SlotIndex.

EC256R1 = 1

RSA = 2

FIDO = 3

class cryptnox_sdk_py.**Derivation**(**values*)
Bases: IntEnum

Predefined values to use for parameters as Derivation.

CURRENT_KEY = 0

DERIVE = 1

DERIVE_AND_MAKE_CURRENT = 2

PINLESS_PATH = 3

class cryptnox_sdk_py.**KeyType**(**values*)
Bases: IntEnum

Predefined values to use for parameters as KeyType.

K1 = 0

R1 = 16

class cryptnox_sdk_py.**AuthType**(**values*)
Bases: Enum

Predefined values for authentication type.

NO_AUTH = 0

PIN = 1

USER_KEY = 2

class cryptnox_sdk_py.**SeedSource**(**values*)
Bases: Enum

Predefined values for how seed was created

NO_SEED = 0

SINGLE = 75

EXTENDED = 88

EXTERNAL = 76

INTERNAL = 83

DUAL = 68

WRAPPED = 82

class cryptnox_sdk_py.**Origin**(**values*)
Bases: Enum

Predefined values for keeping the origin of the card

UNKNOWN = 0

ORIGINAL = 1

FAKE = 2

The cryptnox_sdk_py package is a library for communicating with Cryptnox cards. It exports:

class cryptnox_sdk_py.**Card**
Main card interface class. Alias for *cryptnox_sdk_py.card.base.Base*.

class cryptnox_sdk_py.**Connection**
Connection handler for card communication. See *cryptnox_sdk_py.connection.Connection* for details.

Factory module for creating card instances. See *cryptnox_sdk_py.factory* for details.

Enumeration types module. See *cryptnox_sdk_py.enums* for details.

Exception classes module. See *cryptnox_sdk_py.exceptions* for details.

class exceptions.**SlotIndex**
Card slot index enumeration. See *cryptnox_sdk_py.enums.SlotIndex* for details.

class exceptions.**Derivation**
Key derivation method enumeration. See *cryptnox_sdk_py.enums.Derivation* for details.

class exceptions.**KeyType**
Cryptographic key type enumeration. See *cryptnox_sdk_py.enums.KeyType* for details.

class exceptions.**AuthType**
Authentication type enumeration. See *cryptnox_sdk_py.enums.AuthType* for details.

class exceptions.**SeedSource**
Seed source enumeration. See *cryptnox_sdk_py.enums.SeedSource* for details.

class exceptions.**Origin**
Origin enumeration. See *cryptnox_sdk_py.enums.Origin* for details.

exceptions.**__version__: str = "1.0.4"**
Current version of the cryptnox_sdk_py library.

3 Class Diagrams

Automatically generated visual documentation of the Cryptnox SDK architecture

This section provides automatically generated class diagrams for the Cryptnox SDK Python package. These diagrams are generated directly from the source code and update automatically when the code changes.

What you'll find here:

- **Class Hierarchies** - Inheritance relationships between card, exception, and enum classes
- **System Architecture** - High-level component interactions and data flows
- **Connection Patterns** - Reader and connection class structures
- **Process Flows** - Card initialization and operation sequences

All diagrams are interactive SVG graphics that update automatically with code changes.

3.1 Overview

The Cryptnox SDK follows an object-oriented design with a clear class hierarchy. The diagrams below illustrate the relationships between classes, inheritance structures, and key components.

3.2 Card Class Hierarchy

The main card classes follow an inheritance pattern with a base abstract class and specific implementations.

3.3 Complete Card Module

Complete class hierarchy for all card-related classes:

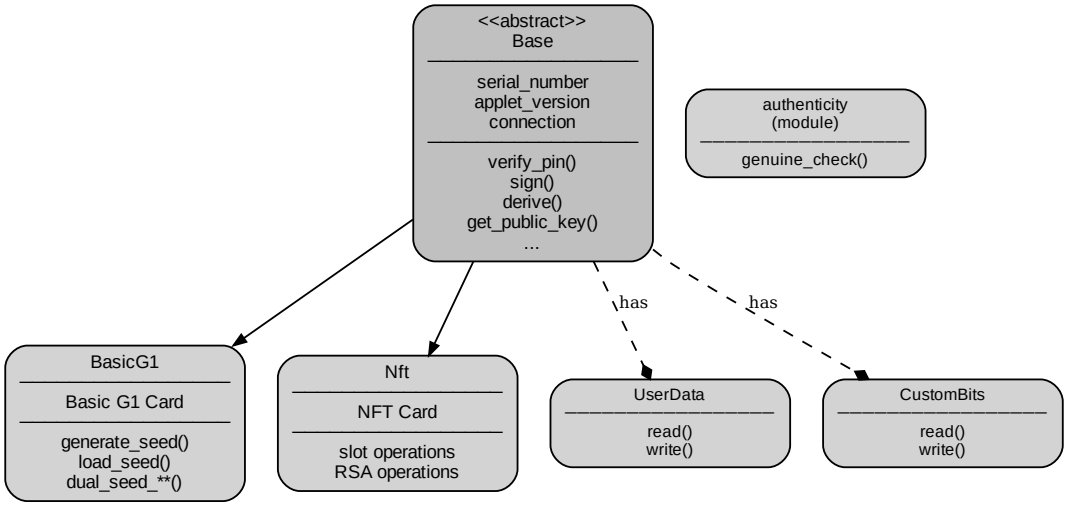

Fig. 1: Complete card module class hierarchy

3.4 Exception Hierarchy

The SDK defines a custom exception hierarchy for different error scenarios:

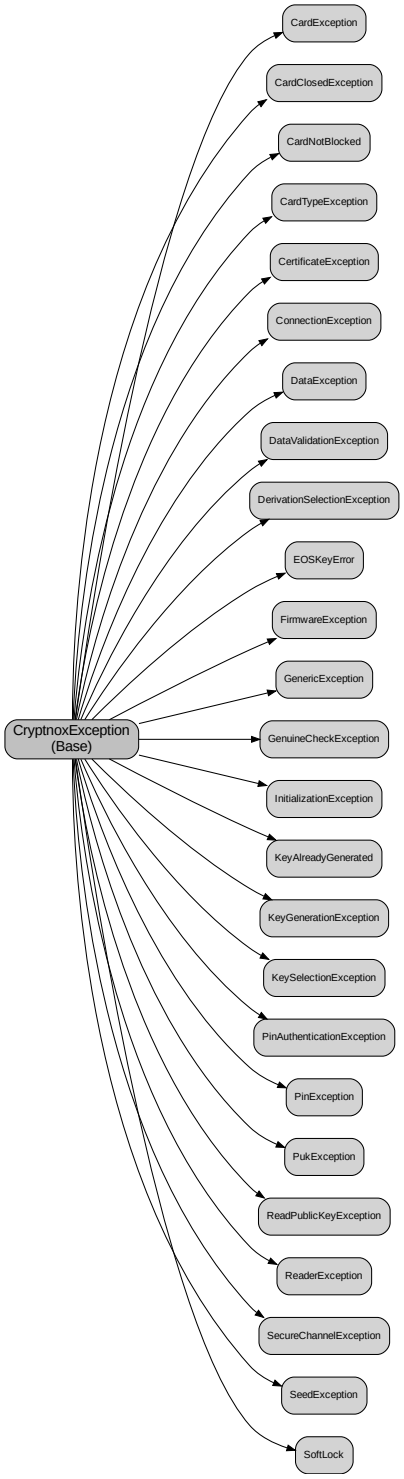

Fig. 2: Exception class hierarchy (expanded view)

3.5 Enum Classes

The SDK uses several enumerations for type safety:

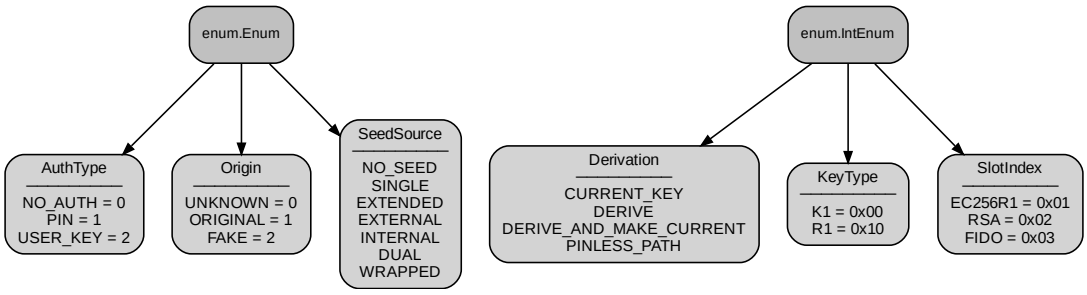

Fig. 3: Enumeration classes

3.6 Connection Components

Classes related to card connection and communication:

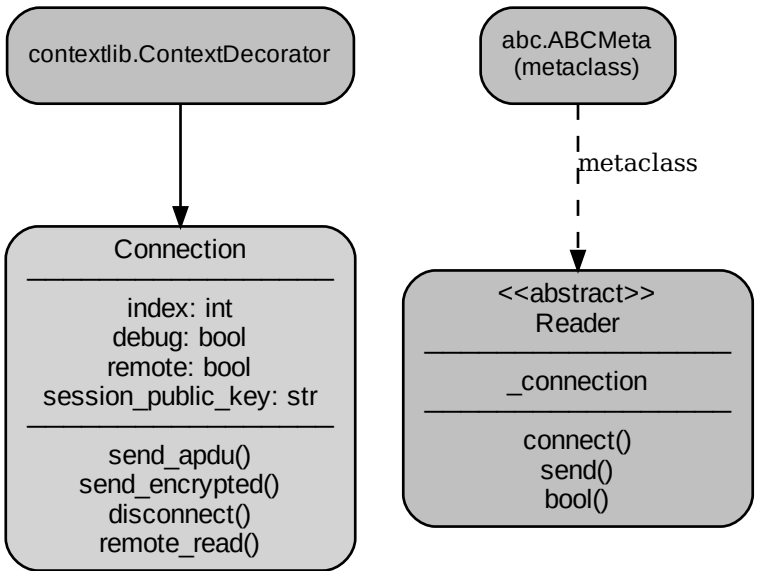

Fig. 4: Connection and Reader classes

3.7 Custom Architecture Diagram

The following diagram shows the high-level architecture of the SDK:

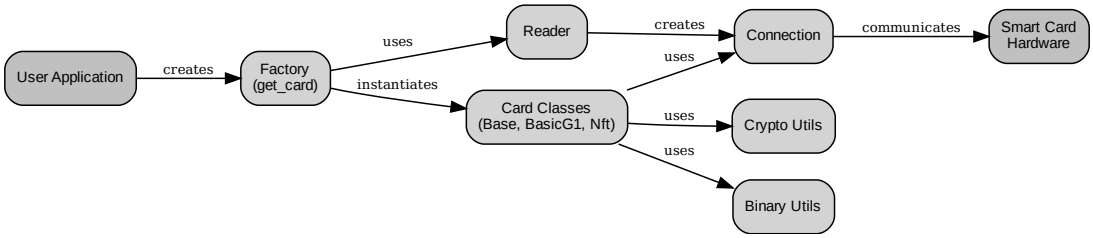

Fig. 5: Cryptnox SDK Architecture

3.8 Data Flow Diagram

The following diagram illustrates the data flow during card operations:

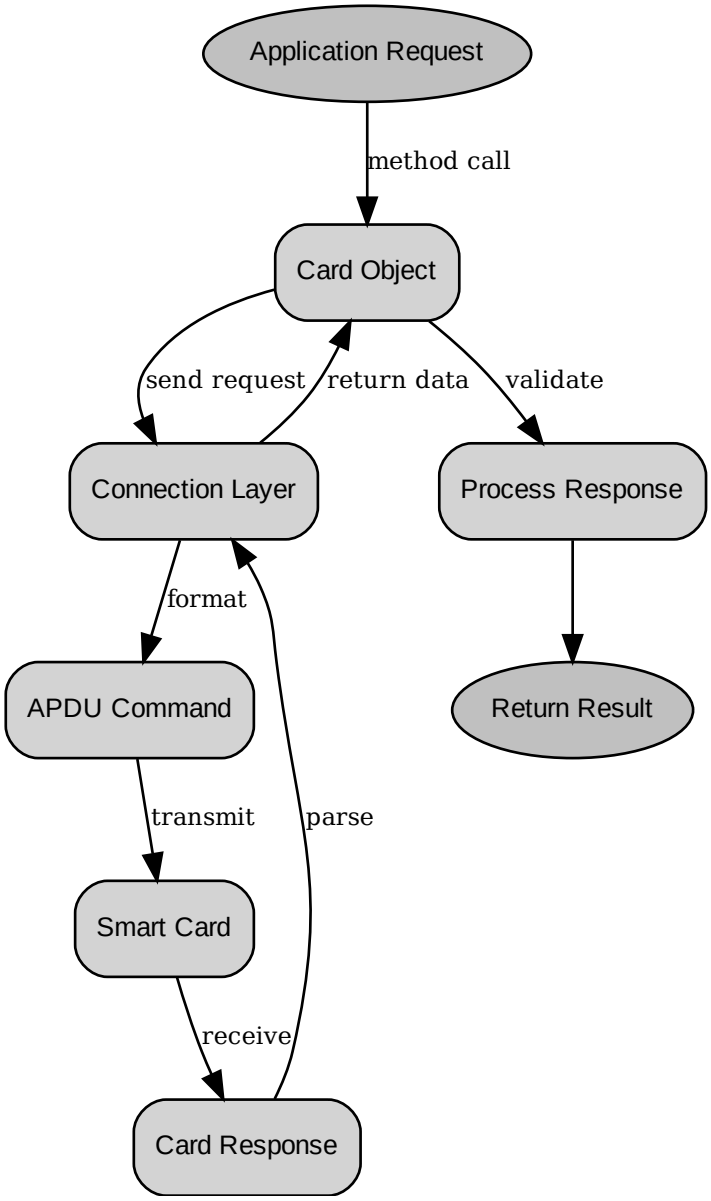

Fig. 6: Card Operation Data Flow

3.9 Card Initialization Sequence

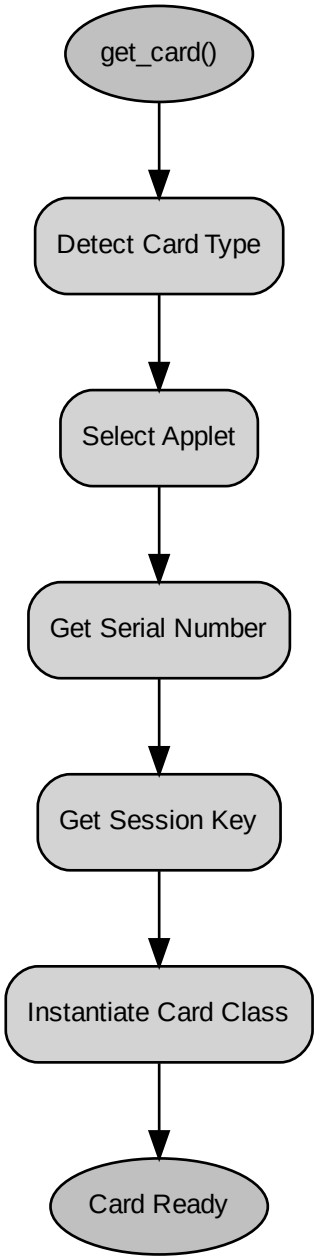

Fig. 7: Card Initialization Process

3.10 Reader Class Hierarchy

The SDK supports different types of card readers:

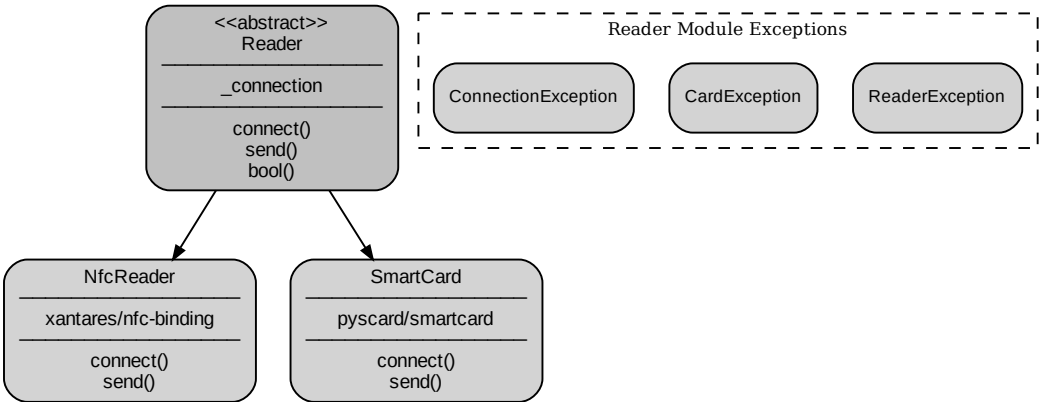

Fig. 8: Reader implementations (NfcReader and SmartCard reader)

3.11 Notes on Diagram Generation

Automatic Updates: All diagrams on this page are generated automatically from the Python source code during the Sphinx build process. When you modify the code structure, simply rebuild the documentation to see updated diagrams.

Technologies Used:

- **Sphinx**: Documentation generator
- **sphinx.ext.inheritance_diagram**: For class hierarchy diagrams
- **sphinx.ext.graphviz**: For custom architecture and flow diagrams
- **Graphviz**: Graph visualization software

Build Requirements: Make sure Graphviz is installed on your system and available in your PATH. See the documentation guides for detailed setup instructions.

3.12 For Developers

If you're a developer working on this project and need to regenerate or customize diagrams, please refer to:

- **Developer Guide**: *docs/DEVELOPER_GUIDE_DIAGRAMS.md* - Complete documentation guide

Quick rebuild command:

```
cd docs
sphinx-build -b html . _build/html
```

4 License

This project is available under the terms of the GNU Lesser General Public License (LGPL) v3.

```
                   GNU LESSER GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.


  This version of the GNU Lesser General Public License incorporates
the terms and conditions of version 3 of the GNU General Public
License, supplemented by the additional permissions listed below.

  0. Additional Definitions.

  As used herein, "this License" refers to version 3 of the GNU Lesser
General Public License, and the "GNU GPL" refers to version 3 of the GNU
General Public License.

  "The Library" refers to a covered work governed by this License,
other than an Application or a Combined Work as defined below.

  An "Application" is any work that makes use of an interface provided
by the Library, but which is not otherwise based on the Library.
Defining a subclass of a class defined by the Library is deemed a mode
of using an interface provided by the Library.

  A "Combined Work" is a work produced by combining or linking an
Application with the Library.  The particular version of the Library
with which the Combined Work was made is also called the "Linked
Version".

  The "Minimal Corresponding Source" for a Combined Work means the
Corresponding Source for the Combined Work, excluding any source code
for portions of the Combined Work that, considered in isolation, are
based on the Application, and not on the Linked Version.

  The "Corresponding Application Code" for a Combined Work means the
object code and/or source code for the Application, including any data
and utility programs needed for reproducing the Combined Work from the
Application, but excluding the System Libraries of the Combined Work.

  1. Exception to Section 3 of the GNU GPL.
```

```
  You may convey a covered work under sections 3 and 4 of this License
without being bound by section 3 of the GNU GPL.

  2. Conveying Modified Versions.

  If you modify a copy of the Library, and, in your modifications, a
facility refers to a function or data to be supplied by an Application
that uses the facility (other than as an argument passed when the
facility is invoked), then you may convey a copy of the modified
version:

   a) under this License, provided that you make a good faith effort to
   ensure that, in the event an Application does not supply the
   function or data, the facility still operates, and performs
   whatever part of its purpose remains meaningful, or

   b) under the GNU GPL, with none of the additional permissions of
   this License applicable to that copy.

  3. Object Code Incorporating Material from Library Header Files.

  The object code form of an Application may incorporate material from
a header file that is part of the Library.  You may convey such object
code under terms of your choice, provided that, if the incorporated
material is not limited to numerical parameters, data structure
layouts and accessors, or small macros, inline functions and templates
(ten or fewer lines in length), you do both of the following:

   a) Give prominent notice with each copy of the object code that the
   Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the object code with a copy of the GNU GPL and this license
   document.

  4. Combined Works.

  You may convey a Combined Work under terms of your choice that,
taken together, effectively do not restrict modification of the
portions of the Library contained in the Combined Work and reverse
engineering for debugging such modifications, if you also do each of
the following:

   a) Give prominent notice with each copy of the Combined Work that
   the Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the Combined Work with a copy of the GNU GPL and this license
   document.
```

```
   c) For a Combined Work that displays copyright notices during
   execution, include the copyright notice for the Library among
   these notices, as well as a reference directing the user to the
   copies of the GNU GPL and this license document.

   d) Do one of the following:

       0) Convey the Minimal Corresponding Source under the terms of this
       License, and the Corresponding Application Code in a form
       suitable for, and under terms that permit, the user to
       recombine or relink the Application with a modified version of
       the Linked Version to produce a modified Combined Work, in the
       manner specified by section 6 of the GNU GPL for conveying
       Corresponding Source.

       1) Use a suitable shared library mechanism for linking with the
       Library.  A suitable mechanism is one that (a) uses at run time
       a copy of the Library already present on the user's computer
       system, and (b) will operate properly with a modified version
       of the Library that is interface-compatible with the Linked
       Version.

   e) Provide Installation Information, but only if you would otherwise
   be required to provide such information under section 6 of the
   GNU GPL, and only to the extent that such information is
   necessary to install and execute a modified version of the
   Combined Work produced by recombining or relinking the
   Application with a modified version of the Linked Version. (If
   you use option 4d0, the Installation Information must accompany
   the Minimal Corresponding Source and Corresponding Application
   Code. If you use option 4d1, you must provide the Installation
   Information in the manner specified by section 6 of the GNU GPL
   for conveying Corresponding Source.)

  5. Combined Libraries.

  You may place library facilities that are a work based on the
Library side by side in a single library together with other library
facilities that are not Applications and are not covered by this
License, and convey such a combined library under terms of your
choice, if you do both of the following:

   a) Accompany the combined library with a copy of the same work based
   on the Library, uncombined with any other library facilities,
   conveyed under the terms of this License.

   b) Give prominent notice with the combined library that part of it
   is a work based on the Library, and explaining where to find the
   accompanying uncombined form of the same work.
```

```
  6. Revised Versions of the GNU Lesser General Public License.

  The Free Software Foundation may publish revised and/or new versions
of the GNU Lesser General Public License from time to time. Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.

  Each version is given a distinguishing version number. If the
Library as you received it specifies that a certain numbered version
of the GNU Lesser General Public License "or any later version"
applies to it, you have the option of following the terms and
conditions either of that published version or of any later version
published by the Free Software Foundation. If the Library as you
received it does not specify a version number of the GNU Lesser
General Public License, you may choose any version of the GNU Lesser
General Public License ever published by the Free Software Foundation.

  If the Library as you received it specifies that a proxy can decide
whether future versions of the GNU Lesser General Public License shall
apply, that proxy's public statement of acceptance of any version is
permanent authorization for you to choose that version for the
Library.
```