Cryptnox Hardware Wallet — API Reference

Release 1.6.1

June 20, 2026

Contents

Doc revision 1.0 — for Cryptnox Hardware Wallet v1.6.1

Basic version with user auth by ECDSA 256r1, RSA2048 and FIDO auth.

This JavaCard Open Platform 3 applet provides basic HSM-like “cryptoki” capabilities for managing crypto-currencies private key secure storage and signature for any blockchain using SECP256k1 or SECP256r1 for ECDSA of the transactions. This applet acts like a hierarchical key tree hardware wallet.

The main purposes are key generation, key derivation, authentication and signing.

Communication with the card happens through a simple APDU interface, together with a Secure Channel guaranteeing confidentiality, authentication and integrity of all commands. Many parts of the card comply with ISO/IEC 7816 (1/2/3/...) and Global Platform standards. T=1 mode used.

The secure channel is quite similar to GP SCP03. The applet uses Extended APDU to stick on T=1 single query, single answer.

The applet hosts a “Dual Basic Group Secret” EC secret key, which is a secret designed to be shared within a pair of Cryptnox Hardware Wallet cards. This is loaded at factory with a one-shot encryption. This secures the dual generation process.

1 Introduction

The Cryptnox Hardware Wallet card is a JavaCard-based secure hardware wallet that provides hardware security module (HSM)-grade protection for blockchain operations. It is engineered to manage cryptographic keys, perform key derivation, and sign transactions directly within the card's secure environment, ensuring that private keys never leave the Secure Element.

The card supports secure storage of seeds and private keys and offers compatibility with both `secp256k1` and `secp256r1` elliptic curves. It is capable of generating ECDSA signatures and Bitcoin Schnorr signatures (BIP340) depending on the selected curve. User authentication mechanisms include PIN, PUK, pairing key, and optional external authentication through a user key or FIDO2 authenticator. These multiple layers of access control ensure both flexibility and robust security.

1.1 Security model

The Cryptnox Hardware Wallet card is built on a Common Criteria EAL6+ certified Secure Element, providing resistance to both physical and logical attacks. All sensitive data exchanges are protected by a Secure Channel protocol, based on principles similar to GlobalPlatform SCP03, which ensures mutual authentication, confidentiality, and integrity through `AES-256` encryption and `CMAC`-based message authentication codes.

Access to card functions is enforced through PIN and PUK-based control, with retry limits and delay mechanisms to prevent brute-force attacks. In addition to local authentication, users can optionally employ an external key, such as a TPM, Secure Enclave, or FIDO2 device, to authorize sensitive operations.

The design also supports a dual-card backup generation mode, allowing two cards to share identical seeds securely. This redundancy mechanism guarantees recoverability without ever exporting or revealing the private seed material outside the Secure Element.

Important

Private keys and seeds are generated and stored exclusively within the Secure Element. There is no command or mechanism to extract raw key material from the card.

See also

- *Secure channel* — detailed Secure Channel protocol and session key derivation
- *Authentication* — PIN, PUK, user key, and FIDO2 authentication mechanisms
- *Seed management* — seed generation, dual generation mode, and key source types

1.2 Supported algorithms and curves

The Cryptnox Hardware Wallet card supports a comprehensive suite of asymmetric, symmetric, and hashing algorithms that align with current blockchain and authentication standards.

1.2.1 Elliptic curves

Curve	Usage	Signature types
`secp256k1`	Bitcoin, Ethereum, and other blockchain ecosystems	ECDSA, Schnorr (BIP340)
`secp256r1` (NIST P-256)	Enterprise security, TLS, FIDO2/WebAuthn	ECDSA

1.2.2 Signature algorithms

Scheme	Curve	Notes
ECDSA (canonical low-S)	`secp256k1`, `secp256r1`	DER-encoded, default mode
ECDSA (EOSIO)	`secp256k1`, `secp256r1`	Filtered to fit EOSIO standard
Schnorr (BIP340)	`secp256k1` only	64-byte raw signature

See also

EC signature — signature types, output formats, and pinless signing

1.2.3 Symmetric cryptography

All secure messaging operations use `AES-256 CBC` for encryption and `AES-256 CMAC` for message integrity and authentication within the Secure Channel.

1.2.4 Key derivation

Hierarchical deterministic key derivation is implemented following:

Standard	Curve	Description
`BIP32`	`secp256k1`	Bitcoin HD wallet standard
`SLIP-0010`	`secp256r1`	Alternative derivation for non-k1 curves

Child keys are derived using `HMAC-SHA512` as per standard definitions. The card also supports pinless path derivation, compatible with `EIP-1581`, allowing selective derivation paths to operate without requiring PIN entry.

See also

Key derivation — dual curve support, derivation sources, and parent key caching

1.2.5 Randomness

The Secure Element integrates a hardware True Random Number Generator (TRNG) compliant with `AIS-20` Class `DRG.3`, used for generating seeds, keypairs, and session nonces.

1.2.6 Hash functions

Algorithm	Usage
`SHA-256`	Message digests, PIN/PUK verification, fingerprint computation
`SHA-512`	Session key derivation, `HMAC-SHA512` for BIP32/SLIP10

1.2.7 Standards compliance

Standard	Scope
ISO/IEC 7816 (Parts 1–4)	Smartcard electrical, transmission, and APDU communication
GlobalPlatform SCP03	Secure, authenticated communication between host and card
BIP32, **BIP39**, **BIP44**	Hierarchical deterministic wallet structures and derivation
SLIP10	Key derivation for non-k1 curves
BIP340	Schnorr signature scheme for Bitcoin / `secp256k1`
EIP-1581	Pinless derivation path standard

This compliance ensures compatibility across blockchain environments, enterprise identity frameworks, and security infrastructures.

2 Technical specifications

The Cryptnox Hardware Wallet card integrates advanced hardware and cryptographic capabilities to deliver a secure, high-performance environment for blockchain operations and key management. It is implemented on a certified Secure Element platform and adheres to international smartcard and cryptographic standards.

2.1 Cryptographic capabilities

The Cryptnox Hardware Wallet card supports a broad range of cryptographic functions for blockchain signing, enterprise authentication, and key derivation.

It operates on both `secp256k1` and `secp256r1` elliptic curves, maintaining independent key derivation trees for each. The hierarchical deterministic wallet structure supports up to eight levels of derivation depth as defined in `BIP32` (for `secp256k1`) and `SLIP10` (for `secp256r1`).

The card supports multiple signature schemes, including ECDSA (DER-encoded with canonical low-S enforcement), EOSIO-compatible ECDSA, and `BIP340` Schnorr signatures for Bitcoin use cases.

For symmetric cryptography, the card uses `AES-256 CBC` for data encryption and `AES-256 CMAC` for message integrity. Key derivation follows the `HMAC-SHA512` standard used in `BIP32` and `SLIP10`.

Table 1: Cryptographic summary

Category	Algorithm	Details
Asymmetric (signing)	ECDSA, Schnorr (BIP340)	`secp256k1`, `secp256r1`
Symmetric (encryption)	`AES-256 CBC`	Secure Channel data encryption
Symmetric (integrity)	`AES-256 CMAC`	Message authentication codes
Key derivation	`HMAC-SHA512`	BIP32 / SLIP10 child key derivation
Hashing	`SHA-256`, `SHA-512`	Digests, verification, session keys
Random generation	TRNG (`AIS-20 DRG.3`)	Seeds, keypairs, session nonces

See also

- *EC signature* — signature types and output formats
- *Key derivation* — derivation depth, dual curve support, and caching
- *Seed management* — seed generation and dual generation mode

2.2 Communication protocols

The Cryptnox Hardware Wallet card communicates using ISO 7816 APDUs over the `T=1` protocol, with full support for extended APDUs to handle larger payloads. All sensitive operations are performed through an authenticated Secure Channel, established via ECDH and protected by `AES-256` encryption.

2.2.1 APDU model

The card follows a standard command/response exchange structure:

```
CLA | INS | P1 | P2 | Lc | Data | Le
```

Extended APDUs (>255 bytes) are supported for operations that require large data exchanges.

See also

Setup & channel commands, *Cryptographic commands*, *Data commands*, *User key & configuration commands* — full APDU command reference

2.2.2 T=1 protocol

A block-oriented communication model with built-in error detection and message chaining, ensuring reliable data transfer even during extended or chained APDUs.

2.2.3 Secure channel

The Secure Channel is derived through an ECDH exchange combined with the pairing key, producing session keys via `SHA-512` derivation. These keys protect the session using `AES-256 CBC` (encryption) and `AES-256 CMAC` (integrity), with replay protection managed by nonces and counters.

See also

Secure channel — full protocol description, mutual authentication, and error conditions

2.2.4 Authentication

Access control relies on a multi-tiered authentication framework:

Method	Credential	Description
PIN	4–9 digits	Required before sensitive operations. Three failed attempts trigger a lockout, recoverable via the PUK.
PUK	12 characters	Used for PIN recovery and card reset, with unlimited attempts but enforced delay mechanisms.
Pairing key	32 bytes	Required to open a Secure Channel; recoverable from PUK if lost.
User keys	ECDSA / RSA / FIDO2	External authentication keys uploaded to authorize transactions.
FIDO2	Challenge-response	External authenticator such as a YubiKey, TPM, or Secure Enclave.

See also

Authentication — PIN retry behavior, user key slots, and FIDO authentication details

2.2.5 Query/response model

Communication is strictly one command per response, ensuring predictable and deterministic interactions without concurrency.

2.3 Secure element features

The Secure Element serves as the foundation of the card's security model, designed to protect cryptographic materials and enforce all access control mechanisms within a tamper-resistant environment.

2.3.1 Security certifications

Certification	Scope
Common Criteria `EAL6+`	Hardware and embedded software assurance
`FIPS 140-2/3`	Cryptographic primitives (`AES`, `SHA`, `RNG`)

The architecture is hardened against side-channel attacks (`DPA`/`SPA`) and fault injection attempts, safeguarding against both passive and active threats.

2.3.2 Memory and isolation

Persistent data is stored in secure EEPROM, which is physically and logically isolated from the JavaCard operating system.

Hardware-level protections prevent unauthorized reads, and all write and erase operations include integrity verification routines. This isolation also enforces non-exportability of private keys and seeds. The hierarchical key derivation engine supports up to eight levels of depth, consistent with BIP32 and SLIP10 standards.

2.3.3 Key management

Private keys and seeds never leave the Secure Element. The card supports a dual-seed generation protocol, allowing two cards to securely generate and share identical seeds without ever exposing key material.

Independent key derivation trees exist for both secp256k1 and secp256r1 curves, enabling simultaneous use across blockchain and enterprise contexts.

All cryptographic operations are gated by PIN-based authentication, with optional pinless paths available for predefined derivation slots where repeated authentication is not required.

Important

There is no APDU command or mechanism to export private keys or seeds from the card. Key material is strictly non-extractable by design.

2.3.4 Communication security

Card communication is based on ISO 7816 APDU exchanges using the T=1 protocol, with full support for extended-length APDUs.

A Secure Channel protocol, built on ECDH key exchange and AES-256 encryption, ensures the confidentiality and integrity of all exchanges.

Each session is mutually authenticated, protected with AES-256 CBC for encryption and AES-256 CMAC for integrity, and reinforced with nonce-based replay protection.

2.3.5 Hardware random number generator

The integrated TRNG, compliant with AIS-20 Class DRG.3, provides entropy for all critical security operations including seed creation, keypair generation, and cryptographic session nonces.

This guarantees randomness at the hardware level, essential for maintaining cryptographic unpredictability.

2.4 Hardware and JavaCard platform

The Cryptnox Hardware Wallet card is built on the `NXP JCOP 4.x` Secure Element platform, a JavaCard-based architecture recognized for its robust security and versatility.

It provides dedicated memory partitions for cryptographic key storage and hierarchical key derivation trees, ensuring strict separation between the secure and application layers.

The integrated True Random Number Generator (TRNG) complies with `AIS-20` Class `DRG.3` certification, providing high-entropy randomness for seed generation, keypair creation, and session nonces.

Note

For a complete overview of the card's lifecycle, from `SELECT` through initialization, pairing, and reset, see *Lifecycle management*.

3 Lifecycle management

3.1 Card startup

After receiving the ATR from the card, one must select the applet to use it. The applet AID is `0xA000001000011 2`. This command provides some basic info about the card applet and status flags.

See also

SELECT and *GET MANUFACTURER CERTIFICATE* commands for full details on the startup sequence and response fields.

3.2 Initialization

Right after the installation, the applet is not initialized and the user needs to send some parameters to use the card. The `INIT` command can only be executed on a fresh card; a reset is required if the card was already initialized. Many base parameters changes require a card reset (except PIN/PUK change).

The parameters are sent using “one shot encryption”: the `AES` key is directly the ECDH of the 2 keys (card and computer). In addition to the personal code number, one needs to generate a random common shared key for the card pairing, used later for a fully authenticated and secured channel.

Table 1: Initialization parameters

Parameter	Size	Description
Owner Name	1 + 20B max	Length byte + up to 20 characters
Owner Email	1 + 60B max	Length byte + up to 60 characters
PIN	9B	4–9 digit number, padded with `0x00` to 9 bytes
PUK	12B	12 digits/letters/bytes
Pairing key	32B	Secure Channel first pairing key

The PIN must be 4 to 9 figure number characters (`'0'–'9'`). If the PIN is less than 9 bytes, it must be filled with `0x00` until 9 characters long.

The parameters sent are encrypted and the command payload is:

```
PubKey | IV | EncryptedParams
```

See also

INIT command for the full APDU specification.

3.3 Pairing

After the initialization, the card and the host must share a common secret to be used as an authenticated secure channel. This secret is required any time further to communicate with the card (using a secure channel). The registration of the first common secret (pairing key) is done during the init phase. There is 1 pairing key slot.

The user can use a common public pairing key and only use PIN for authentication. Otherwise, the user host needs to store its pairing key in a safe place. In the case the client would communicate with several cards, the user needs to associate the pairing key with the instance serial number of the card.

Note

If the user loses their pairing info, a stretched hash of the PUK (`SHA256` applied 32 times) can be used as the pairing key (slot `0xFF`).

See also

- *CHANGE PAIRING KEY* command for pairing key management
- *Secure channel* for the full Secure Channel protocol description

3.4 Reset

When the PUK code is sent into the `RESET` command, the card is reset. The reset PUK code is chosen during the initialization stage. The PUK reset code is a 12 characters long string, which allows a full reset of the card.

On top of this 12 characters secret, the reset is also secured by the user pairing key mechanism (if secret). That means that resetting a card requires, on top of the PUK, to know the pairing key to send the command encrypted. Still, the PUK can act as a pairing key on its own.

The reset command erases everything in the card, and makes the card as it was before the initialization.

Important

The PUK tries are unlimited, but there is an internal loop hash that takes time inside the card for each try. Every 12 tries, a power cycle is required to perform new tries.

See also

- cmd-reset for the full APDU specification
- *Status codes reference* for PUK retry behavior details

4 Secure channel

4.1 Opening a secure channel

In a standard use of the applet, just after selecting the applet, the client opens a secure channel using their pairing key for this card.

Just after `SELECT`, the user queries the card certificate and asks the card for its session public key. The session key of the card is provided in a minimalist certificate, which holds a salt provided by the user.

The user generates an EC key pair (`SessionUserPrivateKey` / `SessionUserPublicKey`) for the session and provides the public key part when querying to open the secure channel.

The card responds with a session salt. All parameters of the secure channel are computed using:

```
ECDHXsecret = SessionCardPrivateKey . SessionUserPublicKey (x)   [Card side]
            = SessionUserPrivateKey . SessionCardPublicKey (x)   [Client side]

SessionCardPublicKey  - received in the SELECT command card answer
SessionUserPublicKey  - transmitted to the card in the OPEN SECURE CHANNEL␣
↪command

AESkey | MACkey = SHA512( ECDHXsecret | PairingKey | SessionSalt )
```

When `P1=0xFF`, the pairing key used is `SHA256^32(PUKcode)`. That way, the PUK can be used as a pairing key, and a user losing their pairing key can still use the PUK to access the card.

See also

GET CARD CERTIFICATE, *OPEN SECURE CHANNEL*, and *MUTUALLY AUTHENTICATE* commands for full APDU specifications.

4.2 Mutual authentication

Right after the secure channel is opened, the user shall test it using the `MUTUALLY AUTHENTICATE` command. A successful encryption-decryption full loop means the device and the client have the same pairing key, authenticating each other.

If the authentication fails the card responds with `0x6982`. In this case the `OPEN SECURE CHANNEL` command must be repeated to generate new keys.

See also

Authentication for PIN and user key authentication that occurs after the Secure Channel is established.

4.3 Encrypted APDUs

After a successful `OPEN SECURE CHANNEL` command, all communications between card and client are encrypted.

Note

Only the data fields of C-APDUs are encrypted. `CLA`, `INS`, `P1`, `P2` for C-APDUs are plaintext. This means no sensitive data should be sent in these parameters. Additionally a MAC is calculated for the entire APDU, including the unencrypted fields.

Because R-APDUs can only contain data if their SW is a success or warning status word (`0x9000`, `0x62XX`, `0x63XX`), when the secure channel is open all responses will have SW `0x9000`. The actual SW is always appended at the end of the response data before encryption, which means the client must interpret the last two bytes of the plaintext response as the SW.

An exception to this is SW `0x6982`, which indicates that the Secure Channel has been aborted and as such is returned without any MAC.

4.3.1 Encryption process

To encrypt the data both the card and the client do the following:

1. The data is padded using the `ISO/IEC 9797-1` Method 2 algorithm.
2. The data is encrypted using `AES` in `CBC` mode using the session key.
3. An `AES CBC-MAC` is calculated over the entire APDU data.
4. The data field of the APDU is set to the MAC followed by the encrypted data.

4.3.2 Decryption process

To decrypt the data both the card and the client do the following:

1. The first 16 bytes of the APDU data are the MAC to be verified.
2. The remaining data is decrypted using `AES` in `CBC` mode using the session key.
3. The padding is removed.

The IV used for the encryption is the last seen MAC from the counterpart. This optimizes the number of transmitted bytes and guarantees protection from replay attacks. For the MAC generation, a zero IV is always used.

4.3.3 MAC generation for C-APDUs

Calculated on the concatenation of:

```
CLA INS P1 P2 LC1 LC2 LC3 00 00 00 00 00 00 00 00 00 | encrypted data field
```

The 9-byte long padding does not become part of the data field and does not affect `LCx`.

- `LCx = (Len 00 00)` if len < 256 bytes
- `LCx = (00 LenMSB LenLSB)` if len >= 256 bytes (big endian short)

4.3.4 MAC generation for R-APDUs

Calculated on the concatenation of:

```
LR1 LR2 LR3 00 00 00 00 00 00 00 00 00 00 00 00 | encrypted data field
```

The 13-byte long padding does not become part of the response field. `LRx` is the length of the encrypted response data field and is not transmitted.

- `LRx = (Len 00 00)` if len < 256 bytes
- `LRx = (00 LenMSB LenLSB)` if len >= 256 bytes (big endian short)

4.3.5 Error conditions

1. If a sensitive command is received without an active Secure Channel, the card shall respond with SW `0x6985` (`SW_CONDITIONS_NOT_SATISFIED`).
2. If a MAC cannot be verified the card shall respond `0x6982` and the Secure Channel must be closed.

Warning

When sending very large data commands in the secure channel (much larger than expected), the reply can be illegible and cannot be decrypted. If the command is too large, the MAC-IV chain is broken, since the auth data for the received command data will not be properly read by the card, and any further message will use outdated MAC data.

See also

Status codes reference for a complete reference of all status words, including Secure Channel specific error codes.

5 Authentication

5.1 PIN authentication

The PIN chosen during the initialization needs to be provided after each EC signature, because a signature resets the PIN validity.

The PIN verification is done in a standard way through the secure channel. The PIN must be between 4 and 9 figures long.

5.1.1 Retry behavior

Event	Behavior
Per-session tries	The PIN can be tested **3 times** before a disconnection of the card is required.
After power cycle	3 more tries are available before the PIN state is locked.
PIN locked	Requires unlocking with the PUK via the `UNBLOCK PIN` command.
`0x63C0` response	Returned when tries need a power cycle, even if the PIN is correct.
Persistent `0x63C0`	After power cycle, if the card still throws `0x63C0`, the PIN is blocked.

After a successful signature command, the PIN verification is reset. A PIN verification is valid until the card powers off or a signature is performed.

See also

- *VERIFY PIN* command for the full APDU specification
- *Status codes reference* for PIN and PUK retry counter details

5.2 User key authentication

Another way to authenticate the user is with an `EC256r1` or `RSA` key pair. There is a random challenge to be signed for PIN-like behavior. The public key is registered in the card, and the “blockchain” EC signature can be allowed with a signature from this user key: hash to be signed with the card must be signed with the user key.

The goal of this function is to use the Cryptnox Hardware Wallet card (to make transactions) in conjunction with a key storage such as iOS Secure Enclave Touch ID or a PC TPM, instead of using the PIN.

The PIN check and user authentication is reset after any EC signature (of all the authorized hashes).

This user auth by ECDSA can also be performed by an external FIDO authenticator.

A user key can be uploaded in a slot only once, and requires the `DELETE USER KEY` command if there is already a key in the slot.

5.2.1 Key slots

Slot	Key type	Format
1	`EC 256r1`	X9.62 uncompressed (65 bytes: `04 | X | Y`)
2	`RSA 2048`	256 bytes modulus, exponent must be `65537`
3	FIDO	`CredIDLength(1B) | CredID(up to 64B) | ECpubkey(65B)`

5.2.2 Authentication modes

There are 2 kinds of user auth:

Mode	P1	Description
Challenge-response	`0x01` / `0x02`	Auth for all commands except `SIGN`. The card provides a random challenge, the user signs it.
Auth for sign	`0x00` / `0x10`	Auth the transaction hash. The challenge is one or a set of hashes to be “blockchain” signed. When `P1` is OR’d with `0x10`, the data message is a hash list (1 to 4 hashes).

5.2.3 FIDO authentication

When the slot index is 3, this is a special case for a FIDO signature. The data must include the counter, making the data input:

```
hash(33-97B) | counter(4B) | ECsignature(EC256r1 ASN1)
```

The card checks the signature as a WebAuthn “user verified” message:

```
sha256hash(rp_id) | [0x05] | counter_word | sha256hash(clientData)
```

Where:

- `rp_id` = `"cryptnox.ch"`
- `clientData` = `'{"type": "webauthn.get", "origin": "https://cryptnox.ch", "challenge": "", "clientExtensions": {}}'`

Note

The restriction for FIDO2 user-auth-for-sign is a maximum of 3 hashes can be authorized at once (instead of 4).

5.3 Disabling PIN auth

With the `SET PIN AUTH` command, one can disable the PIN auth and only let the auth performed by user authentication (public keys registered).

Important

When PIN auth is disabled, only user key authentication (slots 1–3) can authorize operations. Make sure at least one user key is registered before disabling PIN auth.

See also

- *ADD USER KEY*, *CHECK USER KEY*, and *SET PIN AUTH* commands for full APDU specifications
- *EC signature* for details on how signing resets authentication state

6 Seed management

The applet manages a 256-bit master secret called the “seed”. This is the BIP32 master seed, and can be externally computed from a mnemonic to a binary seed using BIP39. The key pairs used for ECDSA are then computationally derived from this seed using SLIP10/BIP32 derivation scheme.

Once the card has a seed or a master key, it is not possible to erase or change it, except by performing a reset with the PUK.

Important

The seed is strictly non-extractable. There is no command to read or export the raw seed from the card. The only way to remove it is a full card reset via the PUK.

6.1 Generation

The master seed can be generated in the card using the random number generator in the chip system (AIS-20 class DRG.3). With the use of the GENERATE KEY command, the seed secret never escapes the card protection. If performed the simple way, there is no way to get a backup of this seed.

6.1.1 Dual generation mode

The dual generation mode allows a cross generation between 2 cards which then share a common seed. One card is a backup of the other. The secret seed cannot be output in any way.

The 2 cards generate each a part of the secret, which is mixed with the other. This is secure because it is not possible to read the seed, and it provides a backup, as the 2 cards are identical after a dual generation initialization. Still in this mode, there is no way to extract the seed and put it in another blockchain wallet.

The common seed is SHA256(ECDH). Even the computer doing the transfer exchange cannot compute the seed, protected by ECDH. Similarly, the computer cannot be part of the key exchange, as this requires a signature shared with a secret key, only located within the cards pair.

Dual generation protocol:

```
card 1 : 80D00400          -> PublicKeySigned of the 1st card
card 2 : 80D00400          -> PublicKeySigned of the 2nd card
card 1 : 80D00500 PK2      -> PublicKeySigned of the 2nd card sent
card 2 : 80D00500 PK1      -> PublicKeySigned of the 1st card sent
```

After this, the 2 cards have a common seed. One card is a backup of the other.

See also

LOAD KEY command for the full APDU specification of seed loading and dual generation.

6.1.2 External random entropy

The card can output random data. The query answer is only output data and is not used internally by the card. It is the user's responsibility to compute a mnemonic using the `GENERATE RANDOM` command and then compute the corresponding seed, which can be uploaded in the card using the `LOAD KEY` command.

6.2 Recovery

The JavaCard applet can load a binary seed (or eventually just an EC key pair). Once this seed is loaded in the card, this card now behaves like the one it was backed up from.

Note

Key derivation paths are not backed up and must be identical to retrieve the same key pairs. After loading a seed, use the `DERIVE KEY` command to navigate to the desired path.

See also

- *LOAD KEY* command for the recovery APDU specification
- *Key derivation* for derivation path management after recovery

6.3 Key source types

After loading or generating, the key source info byte (returned by *READ DATA* with `P1=0`, `P2=0`) indicates the origin:

Code	Description
`0x00`	No key (not seeded)
`'K'`	A single `secp256k1` EC pair was loaded
`'X'`	An extended `secp256k1` key was loaded (a `BIP32` node)
`'L'`	An external seed was loaded
`'S'`	Internal seed generated
`'D'`	Seed generated using dual generation mode

7 Key derivation

The card applet is fully compliant with `BIP32` (`secp256k1` curve) and `SLIP10` (`secp256r1` curve), except the maximum depth of derivation from the master key is **8 levels**.

The card stores the present key pair (and its parent), used for signature. This can be changed using the `DERIVE KEY` command, giving a relative path (from the present or parent key pair), or an absolute path (from the master key pair).

See also

Seed management for how the master seed is generated or loaded before derivation can be used.

7.1 Dual curve support

The card can derive with:

Curve	Standard	Usage
`secp256k1`	`BIP32`	Bitcoin, Ethereum, and other blockchain ecosystems
`secp256r1`	`SLIP10`	Enterprise security, TLS, FIDO2/WebAuthn

A flag in `DERIVE KEY`, `GET PUBKEY`, or `SIGN` is used to select one or the other curve. There are 2 separate key trees in the card, one for `secp256r1` and one for `secp256k1`. They are separated, only linked from the same seed.

7.2 Derivation sources

P1 bits 7-6	Source
`00`	Derive from master keys (absolute path)
`01`	Derive from parent keys
`10`	Derive from current keys (relative path)
`11`	Reserved

7.3 Parent key caching

The ability to start derivation from the parent keys allows to more efficiently switch between children of the same key. Note however that only the immediate parent of the current key is cached so one cannot use this to go back in the keys hierarchy.

If no valid parent key is available (e.g., when the current key pair is the master root), the status code `0x6B00` is returned.

Note

Parent key caching is particularly useful for wallet software that needs to iterate over multiple child addresses (e.g., `m/44'/60'/0'/0/0` through `m/44'/60'/0'/0/n`) without re-deriving from the master each time.

7.4 Performance considerations

For ease of use, the user can derive from the root master node key pair (absolute path) at each card startup, or even before each signature. This takes some time. It is better to store intermediate public keys hash and check the status to observe the current key pair in use.

This off-card complex key management is not needed if the signature volume is below one thousand per day.

See also

- *DERIVE KEY* and *GET PUBKEY* commands for the full APDU specifications
- *EC signature* for on-the-fly derivation during signature operations

8 EC signature

The card applet can sign any 256-bit hash provided, using ECDSA with `secp256k1` or `secp256r1` EC parameters. Most blockchain systems use `SHA2-256` to hash the message, but this card applet is agnostic: the signature is performed on a hash provided by the user.

The derivation of the key pair node can also be done using the signature command (relative or absolute). The card derives just before signing; this cannot be used to sign with a different key, and this cannot change the current stored key.

See also

Key derivation for standalone derivation and path management.

8.1 Signature types

P2	Signature type	Description
`0x00`	ECDSA (canonical low-S)	Default mode, DER-encoded `r` and `s` values
`0x01`	ECDSA (EOSIO)	Filtered to fit EOSIO standard
`0x02`	Schnorr (`BIP340`)	Bitcoin Schnorr, `secp256k1` only, 64-byte output

8.2 Ephemeral nonce

The ephemeral `k` used in the ECDSA and Schnorr is random and different for each signature. For ECDSA, this is automatically performed by the Signature function of the underlying `JCOP4` platform, which internally provides a source of high-quality randomness during the signature.

Important

This applet does **not** use RFC 6979 deterministic digital signature generation. The nonce is always sourced from the hardware TRNG.

8.3 ECDSA output format

The signature is encoded as an X9.62 ASN1 DER sequence of two `INTEGER` values, `r` and `s`, in that order:

```
SEQUENCE ::= { r INTEGER, s INTEGER }
```

Additionally, to be compatible with blockchain specific signatures, the `s` part of the signature is always output as “canonical”, changed for `s` to be on the “lower” side (`s` lower than `n/2`).

8.4 BIP340 Schnorr output format

Returns 64 bytes `R|S` = 2 x 256 bits MSB first, as per `BIP340` standard.

The 32-byte nonce rand value is directly provided by a random source in the `JCOP4` platform. Works only with `secp256k1` keys (current, derive, or pinless).

8.5 Pinless signing

`P1=0x03` is specifically designed for payment transactions. It can be executed without Secure Channel (since no sensitive info is transmitted) and does not require PIN authentication.

The current derivation path on the card remains unchanged, but the signing process is performed using the pinless derivation path previously defined using the `SET PINLESS PATH` command.

The pinless path is restricted to the `EIP-1581` prefix (`m/43'/60'/1581'/...`).

Note

Pinless signing is designed for point-of-sale NFC tap scenarios where the card signs a transaction without requiring user interaction beyond the physical tap.

See also

SET PINLESS PATH for configuring the pinless derivation path.

8.6 PIN and auth reset after signing

An EC signature resets the PIN or user key auth. A PIN verification (or user sign auth) must be performed afterwards and before calling any commands which require user auth (or PIN) checked.

If several hashes are granted by user-auth-for-sign, the card expects the signatures to be done in the exact same order as the authentication. The reset of the auth occurs when all signatures are done (up to 4).

After a successful signature session, the user auth is left opened: if a PIN was provided, the PIN stays verified for other commands. If a user key auth was used, the EC auth is opened for other commands.

In case of bad PIN provided, the PIN auth is disengaged.

See also

- *SIGN* command for the full APDU specification
- *Authentication* for PIN and user key authentication details

9 Command overview

The following table provides a summary of all APDU commands supported by the Cryptnox Hardware Wallet card, along with their authentication and security requirements.

9.1 Application & info

Command	Description	Secure Channel	PIN / User Key	PUK
SELECT	Selects the Cryptnox applet.	✗	✗	✗
Get Card Public Key	Retrieves the card factory EC public key.	✗	✗	✗
Get Manufacturer Certificate	Reads Cryptnox X509 manufacturer certificate (paged).	✗	✗	✗
Get Card Certificate	Retrieves ephemeral session certificate (for secure channel setup).	✗	✗	✗

9.2 Initialization

Command	Description	Secure Channel	PIN / User Key	PUK
INIT	Initializes card with PIN, PUK, and pairing key.	✓ One-shot	✗	✓ Set initial
Open Secure Channel	Establishes Secure Channel with pairing key.	✗	✗	`0xFF` fallback
Mutually Authenticate	Confirms Secure Channel integrity with challenge/response.	✗	✗	✗
Change Pairing Key	Updates Secure Channel pairing key.	✓	✗	✗

9.3 User authentication

Command	Description	Secure Channel	PIN / User Key	PUK
Verify PIN	Verifies user PIN, unlocks card for session.	✓	✓ PIN	✗
Change PIN / PUK	Changes PIN or PUK.	✓	✓ PIN or PUK	✓ To change PUK
Unblock PIN	Unblocks PIN with PUK and new PIN.	✓	✗	✓
Add User Key	Stores external user public key (ECDSA, RSA, FIDO).	✓	✓ PIN / User Key	✓ If PIN disabled
Check User Key	Performs challenge-response authentication using user key.	✓	✓ User Key sig	✗
Delete User Key	Deletes a registered user key slot.	✓	✗	✓
Set Pin Auth	Enables/disables PIN auth (forces User Key only).	✓	✗	✓

9.4 Key management

Command	Description	Secure Channel	PIN / User Key	PUK
Load Key	Loads seed, keypair, or performs dual seed generation.	✓	✓	✗
Generate Key	Generates new seed internally.	✓	✓	✗
Generate RSA Seed Wrap Key	Generates RSA keypair for seed wrapping.	✓	✗	✗
Set Pinless Path	Configures EIP-1581 pinless derivation path.	✓	✗	✓
Set Pub Export	Enables xpub or clear pubkey output.	✓	✗	✓
Get Public Key	Reads current or derived public key, xpub.	✓ Except pinless	✓ Except pinless	✗
Derive Key	Derives new key pair from seed (BIP32 / SLIP-0010).	✓	✓	✗
Generate TRNG Random	Outputs random data (16–64 bytes).	✓	✗	✗

9.5 Signing & decryption

Command	Description	Secure Channel	PIN / User Key	PUK
Sign Public	Certifies the current blockchain public key.	✓	✓	✗
Sign	Signs hash (ECDSA / Schnorr) or data (EdDSA).	✓ Except pinless	✓ Except pinless	✗
Decrypt	ECIES-like decryption / symmetric key output.	✓	✓	✗

9.6 Data & history

Command	Description	Secure Channel	PIN / User Key	PUK
Get Card Info / Read Data	Reads owner info, key source, counters, user slot info.	✓	✓ Protected slots	✗
Get History	Reads signing history slots.	✓	✓	✗
Write Data	Writes user data slot or custom bytes.	✓	✓	✗

9.7 Administration

Command	Description	Secure Channel	PIN / User Key	PUK
Reset	Full factory reset of the card.	✓	✗	✓
Disable Reset	Permanently disables reset capability.	✓	✗	✓

Note

For detailed APDU specifications, parameters, and response codes of each command, refer to the dedicated command pages: *Setup & channel commands*, *Cryptographic commands*, *Data commands*, and *User key & configuration commands*.

10 Setup & channel commands

10.1 SELECT

Request APDU

CLA	INS	P1	P2	LC	Data
0x00	0xA4	0x04	0x00	0x07	AID

Request Data

Field	Size	Description
AID	7B	Application identifier: 0xA0000010000112

The SELECT command is documented in the ISO 7816-4 specifications and is used to select the application on the card, making it the active one. The data field is the AID of the application.

After receiving the ATR from the card, this must be the first command sent. It provides basic information about the card applet status and flags, which the client should use to determine the card's current state and capabilities before proceeding with further operations.

Response Data — 24 bytes

Field	Size	Description
Applet type	1B	'B' = Basic
Applet version	3B	Major.Middle.Minor (e.g. 0x01 0x06 0x00 = v1.6.0)
Status flags	2B	Big-endian short, see below
PubKey flags	2B	Big-endian short, see below
Custom bytes	16B	User-defined, written by WRITE DATA (0xFC P1=1). Default: 0x00

Status flag bits (bit 15 = MSB, bit 0 = LSB):

Bit	Meaning
6	Initialized
5	Seed (master seed or EC pair loaded)
4	Pin Auth possible
3	Pinless enabled
2	xpub output enabled
1	Clear read of current public key enabled

PubKey flag bits:

Bit	Meaning
0	User Public Key #1 (ECDSA 256r1) active
1	User Public Key #2 (RSA 2048) active
2	User Public Key #3 (FIDO2) active

The custom bytes can be used to provide personal hints about how to authenticate with the card, or what's stored inside the card. Note that whoever has the card can freely and easily read this data.

The card serial ID can be read with the `GET MANUFACTURER CERTIFICATE` command which provides the card certificate signed by the manufacturer. The card serial is the serial of the provided X509 certificate.

Status Words

SW	Description
`0x9000`	Success

10.2 GET CARD PUBKEY

Request APDU

CLA	INS	P1	P2	LC	Data
`0x80`	`0xF4`	`0x00`	`0x00`	—	None

Response Data — 65 bytes

Field	Size	Description
Card public key	65B	EC R1 uncompressed point (`04 | X | Y`)

This command reads the card's static EC R1 public key. During the installation of the applet, the card initializes its own EC R1 key pair. This command is only useful for the manufacturer at factory to send the Cryptnox certificate in the card, which contains this card public key, thus authenticating the genuineness of the card.

10.3 GET MANUFACTURER CERTIFICATE

Request APDU

CLA	INS	P1	P2	LC	Data
0x80	0xF7	0x00	Page	—	None

P2 values

P2	Description
0x00	First 253 bytes of certificate + 2 bytes length prefix
0x01+	Remaining pages, 255 bytes each

Response Data

Field	Size	Description
Certificate length	2B	Big-endian short (P2=0 only)
Certificate data	up to 253B/255B	X509 certificate fragment

This command reads the Cryptnox card X509 certificate. This command should be used after each `SELECT` and before the `OPEN SECURE CHANNEL` command in order to:

- Read the Card Serial Number. This is useful to get the right pairing key associated with the card, especially when the user client communicates with several cards and needs to associate the PairingKey with the instance SN.
- Read the card public key, to further check the ECDH card public key authenticity during the secure channel setup.
- Check if the card is genuine by verifying the certificate against the Cryptnox public key.

The answer is limited to 255 bytes per page. `P2=0` returns the first 253 bytes of the certificate prepended with 2 bytes of its total length. `P2>0` returns the remaining parts, paginated by 255 bytes.

Status Words

SW	Description
0x9000	Success

10.4 GET CARD CERTIFICATE

Request APDU

CLA	INS	P1	P2	LC	Data
`0x80`	`0xF8`	`0x00`	`0x00`	`0x08`	Nonce

Request Data

Field	Size	Description
Nonce	8B	Random nonce for certificate freshness

This command generates and provides the card basic certificate. It should be used after each `GET MANUFACTURER CERTIFICATE` and before the `OPEN SECURE CHANNEL` command in order to securely get the card ephemeral public key for the channel encryption key exchange.

The nonce is provided by the command query, so the certificate can only be refreshed and live generated by the card. This ensures that the session public key is fresh and has not been replayed from a previous session.

Response Data — variable (~138-140 bytes)

The basic card certificate is a special compact format:

Field	Size	Description
Tag	1B	`'C'` (0x43)
Nonce	8B	Echo of the provided nonce
Session public key	65B	Ephemeral EC 256r1 card key for ECDH
Signature	70-72B	ASN1-DER ECDSA signature over the above fields, signed with the card private key

The session public key is the ephemeral EC 256r1 card key for the ECDH secure channel establishment. The signature is made from the concatenation of the tag, nonce and session public key, signed with the card's static private key (whose public key is in the Cryptnox certificate). This allows the client to verify the session key authenticity.

Status Words

SW	Description
`0x9000`	Success
`0x6984`	Nonce length is not 8 bytes

10.5 INIT

Request APDU

CLA	INS	P1	P2	LC	Data
`0x80`	`0xFE`	`0x00`	`0x00`	var	PubKey(LV) | IV | Encrypted payload

Preconditions: Card must be in pre-initialized state.

This command is only available when the applet is in pre-initialized state and successful execution brings the applet into the initialized state. This command is needed to allow securely storing secrets on the applet. Currently these are the PIN, PUK and pairing password. The user can also enter personal info (name and email).

The client must take the public key received after the `GET CARD CERTIFICATE` command, generate a random keypair and perform R1 EC-DH to generate an AES key. It must then generate a random IV and encrypt the payload using AES-CBC with ISO/IEC 9797-1 Method 2 padding.

This scheme guarantees protection against passive MITM attacks. Since the applet has no "owner" before the execution of this command, protection against active MITM cannot be provided at this stage. However since the communication happens locally (either through NFC or contacted interface) the realization of such an attack at this point is unrealistic.

Request Data (wire format)

Field	Size	Description
PubKey length	1B	Length of the EC public key (65)
PubKey	65B	Client ephemeral EC R1 uncompressed point for ECDH
IV	16B	AES-CBC initialization vector
Encrypted payload	var	AES-CBC encrypted parameters (ISO 9797-1 M2 padded)

Decrypted payload content

The payload is the concatenation of the User data, PIN, PUK and the Secure Channel first pairing secret. The User data consists of the User Name and the User Email, each prepended with a byte indicating their length. For example: `0x07 | 'Cryptnox' | 0x14 | 'contact@cryptnox.com'`.

These user fields can carry whatever information the user wants to input in the card: their name, an ID number, a key identifier, etc.

Field	Size	Description
Name length	1B	Length of owner name (0-20)
Name	0-20B	Owner name
Email length	1B	Length of owner email (0-60)
Email	0-60B	Owner email
PIN	9B	4-9 digit PIN, right-padded with `0x00`
PUK	12B	PUK code
Pairing secret	32B	First secure channel pairing key

The PIN must be 4 to 9 figure number characters (`'0'-'9'`). If the PIN is less than 9 bytes, it must be filled with `0x00` until 9 chars long.

After successful execution, this command cannot be executed anymore (except after a full reset with PUK). The regular SecureChannel (with pairing) is active and PIN and PUK are initialized and validated.

Status Words

SW	Description
`0x9000`	Success
`0x6D00`	Applet already initialized
`0x6A80`	Invalid data (bad pubkey, non-digits in PIN, wrong decrypted length)
`0x6984`	Decryption invalid (wrong encryption key or bad padding)

10.6 OPEN SECURE CHANNEL

Request APDU

CLA	INS	P1	P2	LC	Data
`0x80`	`0x10`	Key idx	`0x00`	`0x41`	Client session public key

This APDU command is the first step to establish a Secure Channel session. A session is aborted when the application is deselected, either directly or because of a card power off or tear away.

In a standard use of the applet, just after selecting the applet, the client opens a secure channel using their PairingKey for this card. The user generates an EC key pair for the session and provides the public key part in this command.

P1 values

P1	Description
`0x00`	Pairing key slot 0
`0xFF`	Use `SHA256^32(PUK)` as pairing key

When `P1=0xFF`, the PUK can be used as a pairing key. This is useful when the user has lost their pairing info: the PUK hashed 32 times with SHA256 can always be used as a pairing key.

Request Data

Field	Size	Description
Client session public key	65B	EC 256r1 uncompressed point (`04 | X | Y`)

Response Data — 32 bytes

The card generates a random 256-bit salt which is sent to the client.

Field	Size	Description
Session salt	32B	Random salt for key derivation

Key derivation — performed identically by both card and client:

```
ECDHsecret = PrivateKey . CounterpartPublicKey (x-coordinate)

For the card:  ECDHsecret = SessionCardPrivateKey . SessionUserPublicKey (x)
For the user:  ECDHsecret = SessionUserPrivateKey . SessionCardPublicKey (x)

AESkey | MACkey = SHA512( ECDHsecret | PairingKey | Salt )
[0..31]  = AESkey  (256-bit encryption key)
[32..63] = MACkey  (256-bit MAC key)
```

The `SessionCardPublicKey` is received in the `GET CARD CERTIFICATE` response. The `SessionUserPublicKey` is transmitted to the card in this command.

For the mutual authentication, any IV can be used for the first encrypted message. Then the IV to use is the latest MAC field received from the counterpart.

Status Words

SW	Description
`0x9000`	Success
`0x6A86`	Invalid P1 (unknown pairing key index)
`0x6A80`	Data is not a valid public key

10.7 MUTUALLY AUTHENTICATE

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
0x80	0x11	0x00	0x00	var	MAC | Encrypted random

This APDU command allows both parties to verify that the keys generated in the OPEN SECURE CHANNEL step are matching and thus guarantee authentication of the counterpart.

The data sent by both parties is a 256-bit random number. The APDU data is sent encrypted with the keys generated in the OPEN SECURE CHANNEL step. Each party must verify the MAC of the received APDU. If the MAC and padding can be verified, it means that both parties are using the same keys.

Only after this step has been executed can the secure channel be considered to be opened and other commands sent.

Request Data (plaintext before encryption)

Field	Size	Description
Random	32B	256-bit random number

Response Data (plaintext after decryption) — 32 bytes

Field	Size	Description
Random	32B	256-bit random number from the card

If the authentication fails the card responds with 0x6982. In this case the OPEN SECURE CHANNEL command must be repeated to generate new keys.

Note

Once the secure channel is opened, for commands requiring a secure channel, the status error codes are sent back as encrypted data through the tunnel. The last 2 bytes of the decoded data are the command status word, and the “outer plain” data are always 0x9000. The real status word is the one being encrypted in the secure channel inner data payload.

Status Words

SW	Description
0x9000	Success
0x6985	Previous APDU was not OPEN SECURE CHANNEL
0x6982	Authentication failed or data is not 256-bit

10.8 CHANGE PAIRING KEY

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xDA`	`0x00`	`0x00`	var	MAC | Encrypted data

Preconditions: Secure Channel must be opened.

Update the SecureChannel PairingKey for a new one. The PUK is required for authorization.

In case the current pairing key was lost, preventing the opening of any secure channel (needed to change it), one has to use the PUK derived key as the pairing key (key index `0xFF` in `OPEN SECURE CHANNEL`). The PUK pairing slot can't be changed this way, but it is automatically updated when the PUK is changed.

Request Data (plaintext)

Field	Size	Description
New PairingKey	32B	New pairing secret
PUK	12B	Current PUK for authorization

Total plaintext: 44 bytes.

Status Words

SW	Description
`0x9000`	Success
`0x6700`	Data is not 44 bytes
`0x63Cx`	Wrong PUK (x = remaining tries before power cycle)
`0x6A86`	P1 is not 0

11 Cryptographic commands

11.1 VERIFY PIN

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0x20`	`0x00`	`0x00`	var	MAC | Encrypted PIN (or empty)

Preconditions: Secure Channel must be opened.

The PIN verification is done in a standard way through the secure channel. The PIN chosen during the initialization needs to be provided after each EC signature, because a signature resets the PIN validity.

If the PIN entry is correct, the card returns `0x9000` and the PIN is marked as authenticated for the entire session (until the application is deselected, the card is reset/teared, or a signature is performed).

When there is no data in the command, the card returns 1 byte indicating the number of PIN retries left. This allows the client to check the PIN status without attempting a verification.

The PIN must be between 4 and 9 figures long.

Request Data — Verify mode (plaintext)

Field	Size	Description
PIN	4-9B	Digit characters `'0'-'9'`

Request Data — Query mode: Send with empty data.

Response Data (query mode) — 1 byte

Field	Size	Description
Retries remaining	1B	Number of PIN attempts left

Retry behavior: The PIN can be tested 3 times before a disconnection of the card is required. Then it can be tested 3 more times before the PIN state is locked, requiring unlocking with PUK. When the number of tries reaches 3 (per session), the card needs a power cycle before accepting new PIN verification, and it returns `0x63C0`. After a power cycle, if the card still throws `0x63C0`, that means the PIN is blocked and the `UNBLOCK PIN` command must be used.

Status Words

SW	Description
0x9000	PIN correct (verified for session)
0x63Cx	Wrong PIN. x = remaining attempts
0x63C0	Session tries exhausted (power cycle required) or PIN blocked
0x6700	Incorrect data length
0x6986	Card not initialized or PIN auth disabled

11.2 CHANGE PIN

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
0x80	0x21	Selector	0x00	var	MAC | Encrypted data

Preconditions: Secure Channel must be opened. To change the PIN: user PIN must be verified (else PUK must be provided in data).

Change the PIN or PUK. In case of invalid format, the code 0x6A80 is returned. If the conditions match, the PIN or PUK is updated and 0x9000 is returned.

The new PIN must be 9 chars long, must start with at least 4 figure characters ('0'-'9'). If the PIN is less than 9 numbers, it must be filled with 0x00 until 9 chars long. The PUK must be 12 bytes long.

P1=0x00 — Change PIN (PIN already verified):

Field	Size	Description
New PIN	9B	4-9 digits, right-padded with 0x00

The current PUK added for auth in data is mandatory if PinAuth is disabled, as the PIN can't be verified. The new PIN is verified for the session (until card power cycle or EC signature), except if PinAuth is disabled.

P1=0x00 — Change PIN (PIN not verified, use PUK instead):

Field	Size	Description
New PIN	9B	4-9 digits, right-padded with 0x00
PUK	12B	Current PUK for authorization

P1=0x01 — Change PUK:

Field	Size	Description
New PUK	12B	New PUK code
Current PUK	12B	Current PUK for authorization

Total plaintext: 24 bytes.

P1=other — Returns remaining PIN tries with `0x63Cx`. Resets any user auth.

Status Words

SW	Description
`0x9000`	Success
`0x6A80`	Invalid PIN/PUK format
`0x6985`	PIN not validated or card not initialized. Also returned when bad PUK provided to change PIN (this is considered as PIN not validated).
`0x9840`	PUK given is invalid (when P1=0x01)
`0x63Cx`	Remaining PIN tries (when P1=other)

11.3 UNBLOCK PIN

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0x22`	`0x00`	`0x00`	var	MAC | Encrypted data

Preconditions: Secure Channel opened, PIN must be blocked, PinAuth enabled.

Unblock the user PIN. The data field must contain exactly 21 chars, otherwise SW `0x6A80` is returned. The first 12 bytes are the PUK and the last 9 digits are the new PIN. If the PUK is correct the PIN is changed to the given one, PIN is unblocked and authenticated for the rest of the session. The status code `0x9000` is returned.

The PIN must be 9 chars long, must start with 4 figure characters (`'0'-'9'`). If the PIN is less than 9 numbers, it must be filled with `0x00` until 9 chars long.

Request Data (plaintext) — 21 bytes

Field	Size	Description
PUK	12B	PUK code
New PIN	9B	4-9 digits, right-padded with `0x00`

If the PUK is wrong, the SW `0x63Cx` is sent back; `x` is the remaining tries left before a power cycle is required. PUK tries are unlimited, but only 12 tries can be performed on each power cycle. The card needs to be powered off (and then on) before sending new PUK. Else it throws `0x63C0` without checking the PUK.

Status Words

SW	Description
0x9000	Success (PIN unblocked and authenticated)
0x6A80	Format invalid (not 21 bytes or non-digit PIN)
0x6985	PIN is not blocked or PinAuth disabled
0x63Cx	Wrong PUK. x = remaining tries before power cycle

11.4 LOAD KEY

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
0x80	0xD0	Key type	0x00	var	MAC | Encrypted data

Preconditions: Secure Channel opened, PIN or challenge-response validated.

Load a seed or a key pair. Can also be used for dual generation of the seed (shared with 2 cards paired beforehand). Once a seed is loaded in the card, this card now behaves like the one it was backup from. Be aware that key derivation paths are not backed up and must be identical to retrieve the same key pairs.

The PIN must be at the end of the data, unless a user key auth was performed. PIN when provided is always right-filled with 0x00 up to 9 bytes. Load key checks the last 9 bytes as the PIN in case no user auth by key is valid.

P1 values

P1	Description
0x01	ECC SECP256k1 keypair (k1 only). Key source set to 'K'.
0x02	ECC SECP256k1 extended keypair with chain code (k1 only). Key source set to 'X'.
0x03	Binary seed (BIP39, 16-64 bytes). Master key nodes computed following SLIP10, compatible with BIP32. One master key for k1, one for r1. Key source set to 'L'.
0x04	Dual Gen part 1: initializes the dual generation seed mode and sends back a public key signed with the Basic Group key. Nothing is done about any key or seed loading at this stage.
0x05	Dual Gen part 2: receives the public key from the other card (plus signature). After this, both cards have a common seed. Key source set to 'D'.

P1=0x01 — EC keypair (plaintext)

Field	Size	Description
Tag `0xA1`	1B	Keypair template tag
Template length	1-2B	BER-TLV length
Tag `0x80`	1B	Public key tag (optional, can be omitted)
Public key length	1-2B	BER-TLV length
Public key	65B	EC uncompressed point (optional)
Tag `0x81`	1B	Private key tag
Private key length	1-2B	BER-TLV length
Private key	32B	EC private key scalar
PIN	9B	Right-padded with `0x00` (omit if user auth done)

P1=0x02 — Extended keypair: Same as P1=0x01 with additional Tag `0x82` = chain code (32B).

P1=0x03 — Binary seed (plaintext)

Field	Size	Description
Seed	16-64B	BIP39 binary seed
PIN	9B	Right-padded with `0x00` (omit if user auth done)

Dual generation protocol (P1=0x04 and P1=0x05):

```
card 1 : 80D00400          -> PublicKeySigned of the 1st card
card 2 : 80D00400          -> PublicKeySigned of the 2nd card
card 1 : 80D00500 PK2      -> Send 2nd card's signed key to 1st card
card 2 : 80D00500 PK1      -> Send 1st card's signed key to 2nd card
```

After this, the 2 cards have a common seed `SHA256(ECDH)`. One card is a backup of the other, even if no secrets were leaked externally. Even the computer doing the transfer exchange can't compute the seed (protected by ECDH). The computer can't be part of the key exchange, as this requires a signature with a secret key only located within the cards.

The common seed can be checked by making a derivation path and reading the blockchain public keys of the cards, which should match.

Status Words

SW	Description
`0x9000`	Success. PIN is validated for the session.
`0x63Cx`	Wrong PIN
`0x6A80`	Invalid format, invalid dual signature, or PIN expected (data len < 9B)
`0x6A86`	Invalid P1
`0x6986`	Key/seed already loaded (requires reset)

Note

This command can only be performed once. There's no way to erase the seed except with a full reset using the PUK. After successful loading with PIN, the PIN is validated for

the session. The PIN-less path is reset. Unless a `DERIVE KEY` is sent, a subsequent `SIGN` command will use the master keypair for signature.

11.5 DERIVE KEY

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xD1`	Options	`0x00`	var	MAC | Encrypted path

Preconditions: Secure Channel opened, PIN or challenge-response validated, seed loaded.

This command is used before a signing session to generate a private key according to the BIP32 and SLIP10 specifications. The generated key is used for all subsequent `SIGN` sessions. The maximum depth of derivation from the master key is 8. Any attempt to get deeper results in `0x6A80` being returned.

The BIP32 specifications define a few checks which must be performed on the derived keys. If these fail (mostly using r1 curve), there is a looping mechanism described by the SLIP10 proposal, enforced by the card.

The ability to start derivation from the parent keys allows to more efficiently switch between children of the same key. Note however that only the immediate parent of the current key is cached so you cannot use this to go back in the hierarchy. If no valid parent key is available (e.g. when the current key is the master root), the status code `0x6B00` is returned.

P1 bit fields

Bits	Description
bit 4	`0` = secp256k1; `1` = secp256r1
bits 7-6	`00` = from master; `01` = from parent; `10` = from current

Request Data (plaintext)

Field	Size	Description
Path elements	n × 4B	Sequence of 32-bit big-endian integers (0 to 8 levels). Empty = set master key as current.

Each path element is 4 bytes. For hardened derivation, set bit 31 (OR with `0x80000000`).

A client can perform a `GET PUBKEY` command to get the actual current key path and resume derivation using a different path.

Status Words

SW	Description
0x9000	Success
0x6A80	Invalid format or depth > 8
0x6B00	Derivation from parent but no valid parent cached (e.g. master is current)
0x6985	No seed loaded or PIN not verified

11.6 GENERATE TRNG RANDOM

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
0x80	0xD3	Size	0x00	—	None

Preconditions: Secure Channel must be opened.

Used to generate some random data from the chip internal TRNG (AIS 20 class DRG.3). The given size with P1 shall be between 16 and 64 bytes and must be a multiple of 4 (16, 20, 24, . . . , 64).

This command is only an output, to use the chip secure random number generator. For example to generate a mnemonic in the host machine from a confident random source. It doesn't change any state in the card.

Response Data

Field	Size	Description
Random data	P1 bytes	Raw TRNG output from the chip

Status Words

SW	Description
0x9000	Success
0x6A86	Invalid P1 (not 16-64 or not multiple of 4)

11.7 GENERATE KEY

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xD4`	`0x00`	`0x00`	var	MAC | Encrypted PIN

Preconditions: Secure Channel opened.

Generates a new seed and stores keys completely on card. The state of the card after execution is the same as if a `LOAD KEY` command had been performed. With the use of this command, the seed secret never escapes the card protection. If performed the simple way, there's no way to get a backup of this seed.

The key source is set to `'S'`.

Request Data (plaintext)

Field	Size	Description
PIN	9B	Right-padded with `0x00` (omit if user auth done)

Status Words

SW	Description
`0x9000`	Success. PIN is validated for the session.
`0x63Cx`	Wrong PIN
`0x6986`	Seed/key already loaded (requires reset)
`0x6A80`	PIN expected or bad data

Note

This command can be performed only once after a card reset, as there's no way to erase nor change the current wallet seed key, except with a reset with the PUK. After successful generation with PIN, the PIN is validated for the session.

11.8 SIGN

Request APDU (encrypted, except pinless mode)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xC0`	Key/derive	Sig type	var	MAC | Encrypted data (or plaintext for pinless)

Preconditions: Secure Channel opened (except pinless), PIN or user auth or pinless, key loaded.

The card applet can sign any 256-bit hash provided, using ECDSA with 256k1 or 256r1 EC parameters. Most blockchain systems use SHA2-256 to hash the message, but this card applet is agnostic from this, since the signature is performed on a hash provided by the user.

The ephemeral `k` used in the ECDSA and Schnorr is random and different for each signature. For ECDSA, this is automatically performed by the Signature function of the underlying JCOP4 platform. This applet does not use RFC6979 deterministic digital signature generation.

The derivation of the key pair node can also be done using the signature command (relative or absolute). The card derives just before signing. This can't be used to sign with a different key, and this cannot change the current stored key.

Additionally, to be compatible with blockchain specific signatures, the S part of the signature is always output as "canonical", changed for S to be on the "lower" side (S lower than n/2).

P1 values

P1	Description
`0x00`	Current key (k1)
`0x10`	Current key (r1)
`0x01`	Derive + sign with k1 (OR with bits 7-6 for source)
`0x11`	Derive + sign with r1 (OR with bits 7-6 for source)
`0x03`	Pinless path (k1 only, no SC required)

`P1=0x03` is specifically thought for payment transactions. It can be executed without Secure Channel (since no sensitive info is transmitted) and does not require PIN authentication. The current derivation path on the card remains unchanged, but the signing process is performed using the PIN-less derivation path previously defined using the `SET PINLESS PATH` command.

P2 values

P2	Description
`0x00`	ECDSA with canonical low S
`0x01`	ECDSA with EOSIO canonical filter
`0x02`	Bitcoin Schnorr BIP340 (k1 only)

Request Data — P1=0x00/0x10 (current key, plaintext)

Field	Size	Description
Hash	32B	256-bit hash to sign
PIN	9B	Right-padded with `0x00` (omit if user auth or pinless)

Request Data — P1=0x01/0x11 (derive + sign, plaintext)

Field	Size	Description
Hash	32B	256-bit hash to sign
Path elements	n × 4B	Derivation path (32-bit big-endian integers)
PIN	9B	Right-padded with `0x00` (omit if user auth)

Request Data — P1=0x03 (pinless, plaintext, no SC)

Field	Size	Description
Hash	32B	256-bit hash to sign

Response Data — ECDSA (P2=0x00 or 0x01)

The signature is encoded as an X9.62 ASN1 DER sequence: `SEQUENCE { r INTEGER, s INTEGER }`. For usage on some blockchains, one needs to calculate the recovery ID in addition to the signature. To calculate the recovery ID, apply the same algorithm used for public key recovery from a transaction starting with a recovery ID of 0. If the public key matches, you have found the recovery ID, otherwise increment and try again.

Field	Size	Description
Signature	70-72B	ASN.1 DER X9.62 format

Response Data — BIP340 Schnorr (P2=0x02)

Returns 64 bytes `R|S` = 2x256 bits MSB first, as per BIP340 standard. The 32-byte nonce is provided by a random source in the JCOP4 platform. Works only with k1 keys.

Field	Size	Description
R	32B	Schnorr R component (MSB first)
S	32B	Schnorr S component (MSB first)

PIN behavior: The PIN must be with a 9 numbers fixed length, right-filled with `0x00`. If there was no valid user auth or not in pinless mode, the data length expected is counted with the PIN included. A PIN try is always counted when calling this command with a path data longer than a 9 bytes PIN.

An EC signature resets the PIN or user key auth. That means that a PIN verification (or user sign auth) must be performed afterwards and before calling any commands which require a user auth. If several hashes are granted by user-auth-for-sign, the card expects the signatures to be done in the exact same order as the authentication. The reset of the auth occurs when all signatures are done (up to 4).

After a successful signature session, the user auth is left opened: if a PIN was provided, the PIN stays verified for other commands. If a user key auth was used, the EC auth is opened for other commands. In case of bad PIN provided, the PIN auth is disengaged.

Status Words

SW	Description
0x9000	Success
0x6A80	Data too short or path not multiple of 4
0x6A88	Pinless path not defined (P1=0x03)
0x6985	No key loaded
0x63Cx	Wrong PIN
0x6700	Data length mismatch with expected PIN
0x6B00	Unknown P1 or P2, or P2=2 with P1 not k1

11.9 DECRYPT

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
0x80	0xC4	Mode	0x00	var	MAC | Encrypted data

Preconditions: Secure Channel opened, PIN or user auth, seed loaded.

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 an 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:

```
m/17'/910196630'/2006168372'/332516148'/580566270'
```

The symmetric key is computed as `SHA256(k . PubKey)`, where `k` is the private ECP256r1 key in the “decrypt” key slot.

P1 values

P1	Description
0x00	Output symmetric key
0x01	Decrypt data in card using the derived AES key

P1=0x00 — Output symmetric key (plaintext)

Field	Size	Description
PIN	9B	Right-padded with 0x00 (omit if user auth done)
PubKey	65B	Third-party EC R1 uncompressed point (04 | X | Y)

Total: 74B (with PIN) or 65B (user auth).

Response Data (P1=0x00) — 32 bytes

Field	Size	Description
Symmetric key	32B	`SHA256(ECDH) = IV(16B) | AESkey(16B)`

P1=0x01 — Decrypt in card (plaintext)

When P1 is not 0, the hashed ECDH is used internally as an AES128 key: `IV | Key`, with CBC. The padding is left to the outer system, so data must be padded to 16 bytes alignment (and un-padded by the caller).

Field	Size	Description
PIN	9B	Right-padded with `0x00` (omit if user auth done)
PubKey	65B	Third-party EC R1 uncompressed point
Encrypted data	n × 16B	AES-CBC encrypted data (must be 16-byte aligned)

Response Data (P1=0x01)

Field	Size	Description
Decrypted data	var	Plaintext output

Example usage: Let's call the user Alice and her friend Bob. Alice reads the "encrypt" public key by calling `GET PUBKEY` with derive-from-master and derive-r1 for the SLIP17 path, getting `PKA`. Alice shares this public key with Bob. Bob generates an ephemeral ECr1 key pair `kb/PKB`, computes `key = SHA256(kb . PKA)` to get `IV | AESkey`, encrypts the message with AES-CBC, and shares the ciphertext with Alice along with `PKB`. Alice calls `DECRYPT` with `PIN | PKB | ciphertext` and the card gives out the decrypted data.

The PIN/noPIN case in the data is inferred from the current user key auth status. If a user auth was performed with a user signature (CheckUserKey), this command expects no PIN. If no user auth was validated, this command expects a PIN prepending the data.

The PIN state is changed by this command: if wrong PIN, the auth is reset. If correct, the PIN is validated.

Status Words

SW	Description
`0x9000`	Success
`0x6A80`	Incorrect data length
`0x6985`	No key/seed loaded
`0x63Cx`	Wrong PIN
`0x6982`	Data too large (outside secure channel)

11.10 RESET

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
0x80	0xFD	0x00	0x00	var	MAC | Encrypted PUK

Preconditions: Secure Channel must be opened.

When the PUK code is sent into the RESET command, the card is reset. The reset PUK code is chosen during the initialization stage. The PUK reset code is a 12 characters long string, which allows a full reset of the card. On top of this 12 chars secret, the reset is also secured by the user pairing key mechanism (if secret). That means that resetting a card requires, on top of the PUK, to know the PairingKey to send the command encrypted. Still, the PUK can act as a PairingKey on its own.

The reset command erases everything in the card, and makes the card as it was before the initialization. All data in the card will be reset, with no way to recover (except from seed/mnemonic recovery).

Request Data (plaintext) — 12 bytes

Field	Size	Description
PUK	12B	PUK code

Note

The PUK tries are unlimited, but there's an internal loop hash that takes time inside the card for each try. Every 12 tries, a power cycle is required to perform new tries.

In the case of a power loss during reset, the card can only have its blockchain keys deleted. The card would have the flag "seeded" false and "initialized" true. This would need another reset for a full reset, or the keys need to be generated/loaded again.

Status Words

SW	Description
0x9000	Success
0x63Cx	Wrong PUK (x = remaining tries before power cycle)
0x63C0	Power cycle required before new PUK attempts
0x6A80	Data length is not 12 bytes

12 Data commands

12.1 GET CARD INFO / READ DATA

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xFA`	Mode	Selector	—	None (or MAC only)

Preconditions: Secure Channel must be opened.

This command is the main data-retrieval interface of the card. Depending on the P1/P2 combination, it returns card-level metadata, user key slot information, or arbitrary user data pages. It is used after establishing a secure channel to inspect the card's state and read back stored data.

P1/P2 combinations

P1	P2	Description
`0x00`	`0x00`	Card info (name, email, signature counter)
`0x01-0x`	`0x00`	User key slot info (slot = P1)
`0x00`	`0x01-0x`	User data pages (page = P2-1)

Response Data — P1=0, P2=0 (Card info)

When both P1 and P2 are zero, the card returns general identity and status information that was set during initialization with the `INIT` command. This is useful to display the cardholder's name and email, check whether a key has been loaded, and monitor the signing activity through the signature counter.

Field	Size	Description
Key source info	1B	`0x00` = no key; `'K'` = single k1 pair; `'X'` = extended k1 node; `'L'` = external seed; `'S'` = internal seed; `'D'` = dual generation (see *Seed management*)
Name length	1B	Length of owner name
Name	0-20B	Owner name (as set during `INIT`)
Email length	1B	Length of owner email
Email	0-60B	Owner email (as set during `INIT`)
Signature counter	4B	Big-endian unsigned integer, incremented with each `SIGN`

Response Data — P1=1-3, P2=0 (User key slot)

When P1 is set to a slot index (1–3) with P2=0, the card returns the description and public

key stored in that user key slot. These were originally written with the *ADD USER KEY* command. This requires PIN verification or a successful challenge-response beforehand.

Field	Size	Description
Info text	64B	User-provided description (fixed 64 bytes, as written by `ADD USER KEY`)
Public key	65B or 256B	EC R1 uncompressed (65B for slot 1/3) or RSA modulus (256B for slot 2)

Response Data — P2=1-3 (User data pages)

When P2 is set to a page index (1–3), the card returns the user data stored on that page. Pages are written with the *WRITE DATA* command and can hold up to 1200 bytes each, for a total of 3600 bytes of user-defined storage. This also requires PIN or challenge-response.

Field	Size	Description
User data	0-1200B	Data from the requested page

Status Words

SW	Description
`0x9000`	Success
`0x6985`	Secure channel not opened, or PIN/challenge not validated (required when P1>0 or P2>0)
`0x6B00`	Invalid P1/P2 combination or user data slot out of range

Note

When reading user data (P2>0), the command must be sent using the extended frame format header to receive data larger than 256 bytes. This is required by the ISO 7816 standard for long response payloads, even though the command frame itself is short.

12.2 GET HISTORY

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xFB`	Slot	`0x00`	—	None (or MAC only)

Preconditions: Secure Channel opened, PIN or challenge-response validated.

The card maintains a circular history buffer of the last 149 signing operations. Each entry records the value of the signature counter at the time of signing and the 32-byte hash that

was signed. This allows the host application to audit past transactions and verify that the card's signing history matches expected blockchain activity.

The history slot number is provided in P1 (0 to 148). Slot 0 is the most recent entry.

Response Data — 36 bytes

Field	Size	Description
Signing counter	4B	Big-endian unsigned integer (counter value at time of signing)
Signed hash	32B	The 32-byte hash that was signed in this transaction

Status Words

SW	Description
`0x9000`	Success
`0x6985`	Secure channel not opened or PIN/challenge not performed
`0x6A86`	Invalid P1 (slot number out of range, must be 0–148)

12.3 WRITE DATA

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xFC`	Mode	Page	var	MAC | Encrypted data

Preconditions: Secure Channel opened, PIN or challenge-response validated.

This command writes data to one of two storage areas: the user data pages (P1=0) or the custom bytes field (P1=1) that is returned in every `SELECT` response.

P1=0x00 — Write user data page

The card provides three pages of user-defined storage, each up to 1200 bytes, for a total of 3600 bytes. Pages are indexed by P2 (0 to 2) and must be written sequentially starting from page 0. The total user data length is computed as `(P2 x 1200) + data_size`, meaning that lower pages are considered fully written. The card verifies that previous pages were fully written before allowing a higher page to be written; otherwise it returns `0x6985`.

For example, writing 200 bytes with P2=1 sets the total user data length to 1400 bytes (1200 for page 0 + 200 for page 1). The data can later be read back with the *READ DATA* command.

Field	Size	Description
User data	1-1200B	Data for this page

P1=0x01 — Write custom bytes

The custom bytes are 16 bytes of public data that are returned in every `SELECT` response. They can be used to provide personal hints about how to authenticate with the card, or to indicate what is stored inside. Take care that anyone who has physical access to the card can freely read these bytes.

By default, the custom bytes are set to 16 times `0x00`.

Field	Size	Description
Custom bytes	16B	Public data returned in every `SELECT` response

Status Words

SW	Description
`0x9000`	Success
`0x6985`	PIN not validated, or previous pages not fully written (P1=0)
`0x6700`	Data length incorrect (>1200B for P1=0, or not exactly 16B for P1=1)
`0x6A86`	P1 not 0 nor 1, or P2 out of range (0–2)
`0x6982`	Data too large (outside secure channel frame capacity)

12.4 GET PUBKEY

Request APDU (encrypted, or plaintext for pinless/clear read)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xC2`	Derive	Export	var	MAC | Encrypted path (or empty)

Preconditions: Secure Channel opened, PIN or challenge-response validated (except pinless or clear public key read), key loaded.

This is the primary command for exporting public keys from the card. It can return the current key's public key, the current derivation path, or a full BIP32 extended public key (xpub). It also supports on-the-fly derivation: the card computes the derived key from a given path without changing the card's current derivation state, which is useful when the host needs a public key at a specific path for address generation without affecting the signing key.

P1 values

P1	Description
`0x00`	Current key k1
`0x10`	Current key r1
`0x01`	Derive with k1 (OR with bits 7-6 for derivation source)
`0x11`	Derive with r1 (OR with bits 7-6 for derivation source)

When using derivation (P1 LSB = 1), the source key can be selected by OR'ing P1 with the same derivation source flags as the `DERIVE KEY` command:

- Bits 7-6 = `00`: derive from master
- Bits 7-6 = `01`: derive from parent
- Bits 7-6 = `10`: derive from current

P2 values

P2	Description
`0x00`	Read current derivation path
`0x01`	Read public key
`0x02`	Read extended public key (BIP32 xpub, P1 must be `0x00`)

Request Data — P1=0x01/0x11 (derive, plaintext)

Field	Size	Description
Path elements	n x 4B	32-bit big-endian integers (1 to 8 levels)

When P1=0x00/0x10 with P2=0x00 or P2=0x01, no data is required.

Response Data — P2=0x00 (path)

Field	Size	Description
Path elements	n x 4B	Current derivation path as 32-bit big-endian integers

Response Data — P2=0x01 (public key)

Field	Size	Description
Public key	65B	EC uncompressed point (`04 | X | Y`)

Response Data — P2=0x02 (extended public key)

The xpub export must first be enabled using *SET PUB EXPORT* (P1=0). By default this capability is disabled from factory and after reset.

The response approximately matches the binary serialization format defined in the BIP32 standard. The difference is the fingerprint is given as 32 bytes (the full SHA-256 of the parent key) instead of the standard 4 bytes — the host must apply an external RIPEMD-160 computation and take the first 4 bytes to obtain the final BIP32 fingerprint (except for the master key, which is all zeros). There is no checksum, as the data is not Base58-encoded.

Field	Size	Description
Version	4B	BIP32 version bytes (default: BTC mainnet `0x0488B21E`)
Depth	1B	Derivation depth
Fingerprint	32B	Parent key fingerprint (needs external RIPEMD-160 to get 4B)
Child number	4B	Child index
Chain code	32B	BIP32 chain code
Public key	33B	Compressed public key

The xpub can only be read at depth >= 3, in compliance with the BIP44 standard (the first 3 levels are hardened). The first 4 version bytes can be changed on the fly by the host to match the target blockchain; the card always sends BTC mainnet version bytes by default.

The extended public key is useful for wallet software to derive the last account address levels outside the card, manage address changes for each transaction, compute payment addresses in advance, and scan the address chain — all without requiring the card for each derivation.

Pinless and clear read modes

The public key can be read without PIN or secure channel in two scenarios:

1. **Pinless path**: If a pinless path was set with *SET PINLESS PATH*, and the current derivation path starts with the EIP-1581 prefix (`m/43'/60'/1581'/...`), the public key (P2=1) can be read without authentication. No derivation is allowed in this mode.
2. **Clear public key read**: If enabled via *SET PUB EXPORT* (P1=1), the current public key (P1=0/0x10, P2=1) can be read without PIN or secure channel. This is designed for point-of-sale systems that need to read the card's payment address via NFC tap. No derivation is allowed in this mode.

Note

Using the derive option (P1 LSB=1), the card expects a path in the data. There is no way to derive with a null path within this command, so the master key cannot be read via live derivation. If the master public key is needed, set it as the current key with `DERIVE KEY` and then read the current key.

Status Words

SW	Description
`0x9000`	Success
`0x6985`	Not initialized, no key loaded, PIN not verified, or xpub export disabled
`0x6A80`	Malformed path
`0x6A86`	Invalid P1/P2 combination (e.g. P2=2 with P1 != 0)
`0x6A88`	Pinless path not set up, or clear read not allowed
`0x6986`	xpub requested at depth < 3
`0x6983`	Pinless query but current path not in the allowed range
`0x6700`	Current key requested with data present, or derive requested without data

13 User key & configuration commands

13.1 ADD USER KEY

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xD5`	`0x00`	`0x00`	var	MAC | Encrypted data

Preconditions: Secure Channel must be opened.

This command registers a user public key in one of three slots. User keys provide an alternative authentication mechanism to the PIN: instead of typing a numeric code, the user proves identity by signing a challenge or a transaction hash with their private key. This enables integration with hardware security modules like iOS Secure Enclave, PC TPMs, or external FIDO authenticators.

Each slot accepts a different key type, and once a key is written to a slot, it must be deleted with *DELETE USER KEY* before a new key can be stored in the same slot. The key is accompanied by a 64-byte free-text description that can be read back with the *READ DATA* command (P1=slot index, P2=0).

Request Data — Slot 1 (EC 256r1, plaintext) — 142 bytes

Slot 1 stores an EC secp256r1 public key in X9.62 uncompressed format. This is the standard NIST P-256 curve, widely supported by secure enclaves and TPMs.

Field	Size	Description
Slot index	1B	`0x01`
Info text	64B	User-defined description (fixed 64 bytes, padded if shorter)
Public key	65B	EC 256r1 uncompressed (`04 | X | Y`)
PUK	12B	PUK for authorization

Request Data — Slot 2 (RSA 2048, plaintext) — 333 bytes

Slot 2 stores a 2048-bit RSA public key. Only the modulus is transmitted; the public exponent is fixed at 65537 (`0x010001`) and is not sent.

Field	Size	Description
Slot index	1B	`0x02`
Info text	64B	User-defined description (fixed 64 bytes)
RSA modulus	256B	2048-bit modulus, big-endian. Exponent must be 65537.
PUK	12B	PUK for authorization

Note

RSA commands require the extended frame format header, as the payload exceeds 256 bytes.

Request Data — Slot 3 (FIDO, plaintext) — variable

Slot 3 is a special slot designed for FIDO2/WebAuthn authenticators. In addition to the EC public key, it stores the FIDO credential identifier, which is needed by the host to request an assertion from the external authenticator during the *CHECK USER KEY* flow.

Field	Size	Description
Slot index	1B	`0x03`
Info text	64B	User-defined description (fixed 64 bytes)
CredID length	1B	Length of credential ID (up to 64)
CredID	1-64B	FIDO credential identifier
EC public key	65B	EC 256r1 uncompressed (`04 | X | Y`)
PUK	12B	PUK for authorization

Status Words

SW	Description
`0x9000`	Success
`0x6A80`	Invalid slot index
`0x6700`	Incorrect data length
`0x6985`	PIN not provided
`0x6984`	Invalid public key
`0x6986`	Slot already has a key (delete first)

Note

For the current implementation, EC public keys are not tested for point-on-curve validity when saved. On JCOP 4 platforms, the underlying hardware performs this validation implicitly during cryptographic operations.

13.2 CHECK USER KEY

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xD6`	Mode	`0x00`	var	MAC | Encrypted data

Preconditions: Secure Channel must be opened.

This command authenticates the user using a public key previously registered with *ADD USER KEY*. A successful user key authentication is equivalent to a PIN verification. There are two distinct authentication modes, serving different purposes:

1. **Auth for Sign** (P1=0x00 or P1=0x10): The user signs the transaction hash(es) to authorize a subsequent `SIGN` command. The card verifies the signature and, on success, unlocks the `SIGN` command for exactly those hash(es).
2. **Challenge-Response** (P1=0x01 then P1=0x02): The user requests a random challenge from the card, signs it with their private key, and sends the signature back. On success, this unlocks all commands that normally require PIN (except `SIGN`).

Important

The card cannot be simultaneously unlocked for "PIN" commands and for `SIGN`. When the PIN is not used, management commands must be unlocked via challenge-response (P1=1/2), then the `SIGN` must be unlocked separately via auth-for-sign (P1=0/0x10).

P1=0x00 — Auth for Sign (single hash)

Field	Size	Description
Slot index	1B	`0x01-0x03`
Hash	32B	Transaction hash to authorize
Signature	var	EC 256r1 ASN.1 DER (slot 1/3) or RSA 2048 PKCS#1 (slot 2)

On success (response = `0x01`), the `SIGN` command is unlocked for this specific hash only.

P1=0x10 — Auth for Sign (hash list)

When multiple transactions need to be signed in one session, this mode accepts a list of up to 4 hashes (or 3 for FIDO slot). The signed message includes a count byte followed by the concatenated hashes.

Field	Size	Description
Slot index	1B	`0x01-0x03`
Hash count	1B	Number of hashes (1–4, or 1–3 for FIDO)
Hash list	n x 32B	Hashes to authorize (32B each)
Signature	var	EC 256r1 ASN.1 DER (slot 1/3) or RSA 2048 PKCS#1 (slot 2)

P1=0x01 — Challenge Request

No data required (or MAC only). The card generates a random 256-bit nonce and returns it. This challenge must then be signed by the user and submitted with P1=0x02.

Response Data — 32 bytes

Field	Size	Description
Challenge	32B	Random 256-bit nonce

P1=0x02 — Challenge Response

The user submits the signed challenge to prove they hold the private key. The hash is not included in the data because the card uses the challenge it generated internally.

Field	Size	Description
Slot index	1B	`0x01-0x03`
Signature	var	Signature of the challenge (hash is provided by card internally)

On success (response = `0x01`), this unlocks commands requiring PIN (except `SIGN`).

P1P2=0x0301 — Read FIDO Credential ID

Returns the credential ID associated with the FIDO user key slot (slot 3), which the host needs to request an assertion from the external FIDO authenticator.

Response Data:

Field	Size	Description
CredID length	1B	Length of credential ID
CredID	var	FIDO credential identifier

FIDO Registration and Authentication Flow

The following diagram illustrates the complete FIDO2 user registration and authentication flow between the external FIDO authenticator, the host PC, and the wallet card:

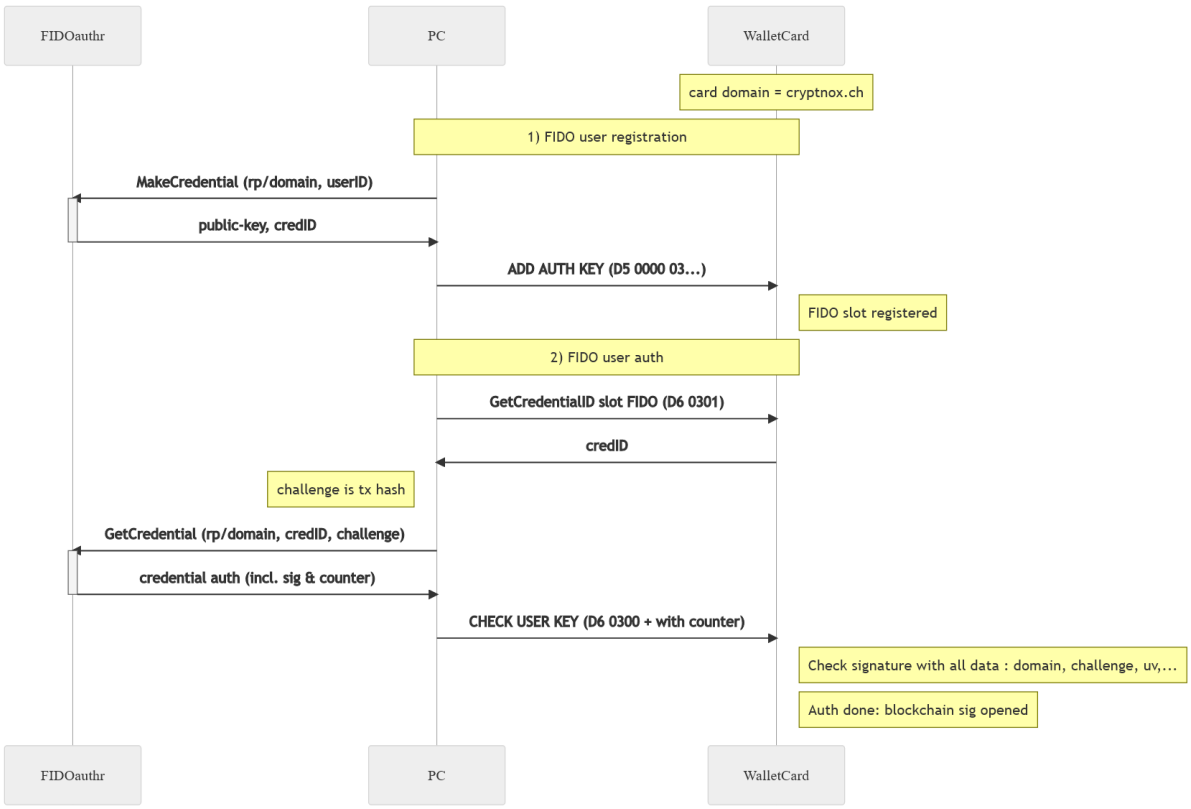

Fig. 1: FIDO2 user registration (ADD USER KEY) and authentication (CHECK USER KEY) sequence.

FIDO Signature Format (Slot 3)

When slot 3 is used for auth-for-sign or challenge-response, the data format differs from EC/RSA slots because it must include the FIDO monotonic counter. The card verifies the signature as a WebAuthn “user verified” assertion:

```
sha256(rp_id) | 0x05 | counter | sha256(clientData)
```

Where `rp_id = "cryptnox.ch"` and `clientData` is a fixed JSON template:

```
{
  "type": "webauthn.get",
  "origin": "https://cryptnox.ch",
  "challenge": "<base64url-encoded hash>",
  "clientExtensions": {}
}
```

FIDO auth-for-sign data format:

Field	Size	Description
Hash(es)	32-97B	Transaction hash(es) to authorize
Counter	4B	FIDO monotonic counter
EC signature	var	EC 256r1 ASN.1 DER

FIDO challenge-response data format (P1=0x02):

Field	Size	Description
Counter	4B	FIDO monotonic counter
EC signature	var	EC 256r1 ASN.1 DER

The FIDO slot is limited to 3 hashes (instead of 4) when using the hash-list mode (P1=0x10).

Response Data (all modes except Challenge Request and CredID read) — 1 byte

Field	Size	Description
Result	1B	`0x01` = success, `0x00` = signature verification failed

Note

A return value of `0x00` with status `0x9000` means the signature was not properly verified: either the signature is invalid or the user key for this slot was not initialized. After a failed challenge attempt, a new challenge must be requested (P1=1) before retrying (P1=2).

Status Words

SW	Description
`0x9000`	Success (check result byte for verification outcome)
`0x6A80`	Invalid slot index or incorrect data
`0x6985`	FIDO key not registered (CredID read), or P1=2 sent before P1=1, or challenge expired (power cycle/deselect)

13.3 DELETE USER KEY

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xD7`	`0x00`	`0x00`	var	MAC | Encrypted data

Preconditions: Secure Channel must be opened.

Deletes the user public key from the specified slot. Once a key has been written to a slot with *ADD USER KEY*, it cannot be overwritten directly — it must first be deleted with this command. The PUK is required for authorization, providing an additional layer of protection against unauthorized key removal.

Request Data (plaintext) — 13 bytes

Field	Size	Description
Slot index	1B	`0x01-0x03`
PUK	12B	PUK for authorization

Status Words

SW	Description
`0x9000`	Success
`0x6A80`	Invalid slot index
`0x6700`	Data length not 13 bytes
`0x63Cx`	Wrong PUK (x = remaining tries before power cycle)
`0x6986`	Slot is empty (nothing to delete)

13.4 SET PINLESS PATH

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xC1`	`0x00`	`0x00`	var	MAC | Encrypted data

Preconditions: Secure Channel opened, seed or extended key loaded.

This command enables the pinless signing feature, which allows the `SIGN` command to be executed without PIN authentication or even a secure channel. This is specifically designed for contactless (NFC) payment transactions where typing a PIN is impractical, such as tap-and-pay at a point of sale.

For security, the pinless path is restricted to a BIP32 derivation path that must begin with the EIP-1581 prefix `m/43'/60'/1581'`. EIP-1581 defines a purpose path for non-wallet usage of keys, providing a clear segregation between pinless keys and standard PIN-protected wallet keys. Despite this naming, the keys can still be used for wallet transactions — the path restriction is purely a security boundary.

The path must be at least 3 levels deep and at most 8 levels. It is provided as raw 32-bit big-endian integers, 4 bytes per level, in the same format as the `DERIVE KEY` command.

This feature only supports the K1 (secp256k1) key pair.

Request Data — Enable pinless (plaintext)

Field	Size	Description
PUK	12B	PUK for authorization
Path elements	12-32B	3-8 levels x 4 bytes (32-bit big-endian)

Path must start with `m/43'/60'/1581'` (EIP-1581 prefix):

```
0x8000002B | 0x8000003C | 0x8000062D | ...
```

Request Data — Disable pinless (plaintext)

To disable the pinless feature, send the command with only the PUK and no path data. The PUK verification still applies.

Field	Size	Description
PUK	12B	PUK for authorization (no path = disable)

Status Words

SW	Description
`0x9000`	Success
`0x63Cx`	Wrong PUK (x = remaining tries before power cycle)
`0x6A80`	Data length not a multiple of 4, or path not between 12 and 32 bytes
`0x6983`	Path doesn't start with EIP-1581 prefix (`m/43'/60'/1581'`)
`0x6985`	No seed or extended key loaded (checked before PUK)

13.5 SET PIN AUTH

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
`0x80`	`0xC3`	`0x00`	`0x00`	var	MAC | Encrypted data

Preconditions: Secure Channel must be opened.

This command controls whether PIN-based authentication is allowed. By default, both PIN and user key authentication are available. Disabling PIN auth forces the user to authenticate exclusively through the *CHECK USER KEY* mechanism (EC/RSA/FIDO signatures).

This is useful in high-security deployments where a numeric PIN is considered insufficient and all authentication must go through a hardware key (Secure Enclave, TPM, FIDO authenticator).

When disabling PIN auth (status byte > 0), the card verifies that at least one user public key has been registered; otherwise it returns `0x6986`. This prevents accidentally locking the user out of the card.

To re-enable PIN auth, call this command with the status byte set to `0x00`.

Request Data (plaintext) — 13 bytes

Field	Size	Description
Status	1B	`0x00` = enable PIN auth; `>0` = disable PIN auth
PUK	12B	PUK for authorization

Status Words

SW	Description
0x9000	Success
0x63Cx	Wrong PUK (x = remaining tries before power cycle)
0x6A80	Data length is not 13 bytes
0x6986	Disable requested but no user public key loaded

13.6 SET PUB EXPORT

Request APDU (encrypted)

CLA	INS	P1	P2	LC	Data
0x80	0xC5	Feature	0x00	var	MAC | Encrypted data

Preconditions: Secure Channel opened, seed loaded.

This command controls two independent public key export features. Both are disabled by default from factory and after every card reset, ensuring that public key data is not accidentally exposed.

P1=0x00 — Extended public key (xpub) export

Enables or disables the ability to read extended public keys (BIP32 xpub) via the *GET PUBKEY* command with P2=0x02. The xpub contains the chain code and is sufficient to derive all child public keys at that level and below, making it a sensitive piece of data that allows address enumeration.

P1=0x01 — Clear public key reading (without PIN or SC)

Enables or disables the ability to read the current public key without PIN verification or even a secure channel. This is designed for point-of-sale scenarios where a reader needs to identify the card's payment address via a simple NFC tap. When enabled, the public key can be read by anyone with physical access to the card.

P1 values

P1	Description
0x00	xpub export capability
0x01	Clear public key reading (without PIN or SC)

Request Data (plaintext) — 13 bytes

Field	Size	Description
Status	1B	0x00 = disable; >0 = enable
PUK	12B	PUK for authorization

Status Words

SW	Description
`0x9000`	Success
`0x63Cx`	Wrong PUK (x = remaining tries before power cycle)
`0x6A80`	Data length is not 13 bytes
`0x6985`	No seed or extended key loaded
`0x6A86`	P1 is not 0 or 1

14 Status codes reference

14.1 Standard status words

SW	Description
`0x9000`	Success. When secure channel is open, this is always the outer SW; the real SW is encrypted in the response data.
`0x63Cx`	Authentication failure. `x` = remaining tries. `0x63C0` = no tries left (power cycle required for PIN, or PUK tries exhausted for this session).
`0x6700`	Wrong data length.
`0x6982`	Security status not satisfied. MAC verification failed — secure channel is aborted. Returned without MAC.
`0x6983`	Path not allowed (e.g., pinless path does not start with `EIP-1581`).
`0x6984`	Invalid data (decryption failed, invalid public key).
`0x6985`	Conditions not satisfied (secure channel not open, PIN not verified, card not initialized, seed not loaded).
`0x6986`	Command not allowed (key slot already filled, PIN auth disabled, xpub at depth < 3).
`0x6A80`	Incorrect data parameters (invalid format, bad PIN format, malformed path).
`0x6A86`	Incorrect `P1`/`P2` parameter.
`0x6A88`	Referenced data not found (pinless path not set, clear pubkey read not allowed).
`0x6B00`	Wrong parameters (invalid parent key, unknown `P1`/`P2` value).
`0x6D00`	Instruction not supported (e.g., `INIT` on already initialized card).
`0x9840`	PUK verification not successful (used in `CHANGE PIN` with `P1=PUK`).

14.2 Secure channel status words

When the secure channel is open, all response status words are `0x9000` in the outer (plaintext) APDU. The actual status word is the last 2 bytes of the decrypted response data.

The only exception is `0x6982`, which indicates the secure channel has been aborted and is returned in plaintext (without MAC).

See also

Secure channel for the full encryption/decryption process and MAC generation details.

14.3 PIN retry behavior

Counter	Behavior
Per-session (transient)	3 tries per power cycle. After 3 failures, returns `0x63C0` until power cycle.
Persistent (`OwnerPIN`)	6 total tries across sessions. Depleted = PIN blocked, requires PUK to unblock.

See also

VERIFY PIN and cmd-unblock-pin for the full APDU specifications.

14.4 PUK retry behavior

Counter	Behavior
Per-session (transient)	12 tries per power cycle. After 12 failures, returns `0x63C0` until power cycle.
Persistent	None. PUK tries are unlimited across power cycles.

See also

Lifecycle management for details on card reset using the PUK.

15 Cryptographic assets

The Cryptnox Hardware Wallet card manages several cryptographic assets throughout its lifecycle. This section provides an overview of all key material, credentials, and derived secrets handled by the card.

15.1 Factory Dual Basic Group Secret

Scope	Secure Element (factory-loaded)
Description	EC secret used for the dual generation protocol and to sign public material exchanged between paired cards. Protects the integrity of the dual-gen exchange.
Notes	Loaded at factory; used in dual generation signatures.

15.2 Card long-term attestation key

Scope	Secure Element (card keypair)
Description	Card's permanent EC keypair (R1) used to sign the card certificate and authenticate ephemeral session keys.
Notes	Readable only as the certificate via `GET MANUFACTURER CERTIFICATE` / `GET CARD CERTIFICATE`.

15.3 Session ephemeral private key

Scope	Secure Element (ephemeral, per session)
Description	Short-lived EC private key generated inside the card and used for ECDH in the secure channel. Never exported.
Notes	Exposed externally only as the card's ephemeral public key inside the basic card certificate (`GET CARD CERTIFICATE`).

15.4 Pairing key

Scope	Card secure storage (32 bytes)
Description	The 32-byte secret used to derive AES/MAC session keys with a host for the authenticated secure channel. There is one pairing key slot; can be public if desired but normally kept secret.
Notes	Set at `INIT`; used in `OPEN SECURE CHANNEL` key derivation; changeable with `CHANGE PAIRING KEY`. Fallback: a PUK-derived pairing key (SHA-256^{32} of PUK) can be used (index `0xFF`).

15.5 Secure Channel session keys (AES / MAC)

Scope	Volatile (session only)
Description	Keys derived from ECDH(SessionCardPriv, SessionUserPub) || PairingKey || SessionSalt via SHA-512, split into AES and MAC keys. Protect confidentiality and integrity of APDUs while the channel is open.
Notes	Derived during `OPEN SECURE CHANNEL`. Not persistent.

15.6 PIN

Scope	Card secure storage
Description	User numeric PIN (4–9 digits) used to authenticate the user for most protected operations. PIN verification state is session-valid until a signature or deselect/power-off.
Notes	`INIT`, `VERIFY PIN`, `CHANGE PIN`, `UNBLOCK PIN`. Retry counters and power-cycle rules apply.

15.7 PUK

Scope	Card secure storage (12 bytes)
Description	Card reset/unblock secret. Used to unblock PIN, authorize PUK-protected changes (e.g., change pairing key, set pinless path, set pub export), and perform `RESET`.
Notes	`INIT`, `RESET`, `CHANGE PAIRING KEY`, `SET PIN AUTH`. Unlimited attempts but throttled with power-cycle behavior.

15.8 PUK-derived pairing fallback

Scope	Derived (fallback)
Description	Deterministic pairing key obtained by hashing the PUK 32 times with SHA-256; usable as pairing key index `0xFF` if pairing information is lost.
Notes	Used via `OPEN SECURE CHANNEL` with `P1=0xFF`.

15.9 User private keys (off-card)

Scope	Host device (e.g., TPM / Secure Enclave)
Description	Private keys kept on a host device used to perform challenge-response authentication against the card. The card stores only the corresponding public key. These act as PIN replacements.
Notes	Card stores public keys in slots via `ADD USER KEY`; verification via `CHECK USER KEY`. Private keys must be protected by the host.

15.10 FIDO credential (slot 3)

Scope	Card slot (credential ID + public key)
Description	FIDO credential identifier and associated EC public key. Used to verify WebAuthn-style signatures for PIN replacement or signing authorization.
Notes	`ADD USER KEY` (slot 3) stores credential ID length, credential ID, EC public key, and PUK. Verified via `CHECK USER KEY`.

15.11 Pinless path derivation settings

Scope	Card secure storage
Description	The BIP32 derivation path used for PIN-less signing (e.g., `m/43'/60'/1581'/...`); guarded by PUK to set/unset. Enables transactions without PIN when used via the pinless `SIGN` mode.
Notes	`SET PINLESS PATH` with PUK. Pinless signing via `SIGN` with `P1=0x03`.

15.12 Dual-generation partial secrets

Scope	Secure Element (temporary during dual-gen)
Description	During the dual-generation protocol, two cards each generate a partial secret and exchange signed public material to produce a shared seed (SHA-256(ECDH)). Each card stores its part; the final shared seed becomes the card seed.
Notes	`LOAD KEY` with `P1=0x04` / `P1=0x05` sequence. Signature checks use the Basic Group Secret.

15.13 Master seed

Scope	Secure Element (non-exportable)
Description	The 256-bit master seed (BIP32 / SLIP-0010) – the root of all derived blockchain keys. This is the ultimate cryptographic asset: it cannot be read or exported, only destroyed via `RESET` with PUK.
Notes	Generated on-card (`GENERATE KEY`) or loaded (`LOAD KEY` with `P1=0x03`). Once set, it cannot be changed except by `RESET` with PUK.

Important

All cryptographic assets stored in the Secure Element are strictly non-exportable. Private keys and seeds never leave the card boundary. Recovery is only possible through dual-generated backup cards or re-importing an externally held seed.

16 License

This documentation is licensed under CC BY-NC-ND 4.0.

- **Attribution** for appropriate credit to Cryptnox SA
- **NonCommercial** for non-commercial use only
- **NoDerivatives** for unmodified redistribution only

For inquiries, contact: contact@cryptnox.com

```
Creative Commons Attribution-NonCommercial-NoDerivatives 4.0
International Public License

By exercising the Licensed Rights (defined below), You accept and agree
to be bound by the terms and conditions of this Creative Commons
Attribution-NonCommercial-NoDerivatives 4.0 International Public
License ("Public License"). To the extent this Public License may be
interpreted as a contract, You are granted the Licensed Rights in
consideration of Your acceptance of these terms and conditions, and the
Licensor grants You such rights in consideration of benefits the
Licensor receives from making the Licensed Material available under
these terms and conditions.


Section 1 -- Definitions.

  a. Adapted Material means material subject to Copyright and Similar
     Rights that is derived from or based upon the Licensed Material
     and in which the Licensed Material is translated, altered,
     arranged, transformed, or otherwise modified in a manner requiring
     permission under the Copyright and Similar Rights held by the
     Licensor. For purposes of this Public License, where the Licensed
     Material is a musical work, performance, or sound recording,
     Adapted Material is always produced where the Licensed Material is
     synched in timed relation with a moving image.

  b. Copyright and Similar Rights means copyright and/or similar rights
     closely related to copyright including, without limitation,
     performance, broadcast, sound recording, and Sui Generis Database
     Rights, without regard to how the rights are labeled or
     categorized. For purposes of this Public License, the rights
     specified in Section 2(b)(1)-(2) are not Copyright and Similar
     Rights.

  c. Effective Technological Measures means those measures that, in the
     absence of proper authority, may not be circumvented under laws
     fulfilling obligations under Article 11 of the WIPO Copyright
     Treaty adopted on December 20, 1996, and/or similar international
```

```
     agreements.

  d. Exceptions and Limitations means fair use, fair dealing, and/or
     any other exception or limitation to Copyright and Similar Rights
     that applies to Your use of the Licensed Material.

  e. Licensed Material means the artistic or literary work, database,
     or other material to which the Licensor applied this Public
     License.

  f. Licensed Rights means the rights granted to You subject to the
     terms and conditions of this Public License, which are limited to
     all Copyright and Similar Rights that apply to Your use of the
     Licensed Material and that the Licensor has authority to license.

  g. Licensor means the individual(s) or entity(ies) granting rights
     under this Public License.

  h. NonCommercial means not primarily intended for or directed towards
     commercial advantage or monetary compensation. For purposes of
     this Public License, the exchange of the Licensed Material for
     other material subject to Copyright and Similar Rights by digital
     file-sharing or similar means is NonCommercial provided there is
     no payment of monetary compensation in connection with the
     exchange.

  i. Share means to provide material to the public by any means or
     process that requires permission under the Licensed Rights, such
     as reproduction, public display, public performance, distribution,
     dissemination, communication, or importation, and to make material
     available to the public including in ways that members of the
     public may access the material from a place and at a time
     individually chosen by them.

  j. Sui Generis Database Rights means rights other than copyright
     resulting from Directive 96/9/EC of the European Parliament and of
     the Council of 11 March 1996 on the legal protection of databases,
     as amended and/or succeeded, as well as other essentially
     equivalent rights anywhere in the world.

  k. You means the individual or entity exercising the Licensed Rights
     under this Public License. Your has a corresponding meaning.


Section 2 -- Scope.

  a. License grant.

       1. Subject to the terms and conditions of this Public License,
          the Licensor hereby grants You a worldwide, royalty-free,
```

```
   non-sublicensable, non-exclusive, irrevocable license to
   exercise the Licensed Rights in the Licensed Material to:

     a. reproduce and Share the Licensed Material, in whole or
        in part, for NonCommercial purposes only; and

     b. produce and reproduce, but not Share, Adapted Material
        for NonCommercial purposes only.

2. Exceptions and Limitations. For the avoidance of doubt, where
   Exceptions and Limitations apply to Your use, this Public
   License does not apply, and You do not need to comply with
   its terms and conditions.

3. Term. The term of this Public License is specified in Section
   6(a).

4. Media and formats; technical modifications allowed. The
   Licensor authorizes You to exercise the Licensed Rights in
   all media and formats whether now known or hereafter created,
   and to make technical modifications necessary to do so. The
   Licensor waives and/or agrees not to assert any right or
   authority to forbid You from making technical modifications
   necessary to exercise the Licensed Rights, including
   technical modifications necessary to circumvent Effective
   Technological Measures. For purposes of this Public License,
   simply making modifications authorized by this Section 2(a)
   (4) never produces Adapted Material.

5. Downstream recipients.

     a. Offer from the Licensor -- Licensed Material. Every
        recipient of the Licensed Material automatically
        receives an offer from the Licensor to exercise the
        Licensed Rights under the terms and conditions of this
        Public License.

     b. No downstream restrictions. You may not offer or impose
        any additional or different terms or conditions on, or
        apply any Effective Technological Measures to, the
        Licensed Material if doing so restricts exercise of the
        Licensed Rights by any recipient of the Licensed
        Material.

6. No endorsement. Nothing in this Public License constitutes or
   may be construed as permission to assert or imply that You
   are, or that Your use of the Licensed Material is, connected
   with, or sponsored, endorsed, or granted official status by,
   the Licensor or others designated to receive attribution as
   provided in Section 3(a)(1)(A)(i).
```

```
  b. Other rights.

       1. Moral rights, such as the right of integrity, are not
          licensed under this Public License, nor are publicity,
          privacy, and/or other similar personality rights; however, to
          the extent possible, the Licensor waives and/or agrees not to
          assert any such rights held by the Licensor to the limited
          extent necessary to allow You to exercise the Licensed
          Rights, but not otherwise.

       2. Patent and trademark rights are not licensed under this
          Public License.

       3. To the extent possible, the Licensor waives any right to
          collect royalties from You for the exercise of the Licensed
          Rights, whether directly or through a collecting society
          under any voluntary or waivable statutory or compulsory
          licensing scheme. In all other cases the Licensor expressly
          reserves any right to collect such royalties, including when
          the Licensed Material is used other than for NonCommercial
          purposes.


Section 3 -- License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the
following conditions.

  a. Attribution.

       1. If You Share the Licensed Material, You must:

            a. retain the following if it is supplied by the Licensor
               with the Licensed Material:

                 i. identification of the creator(s) of the Licensed
                    Material and any others designated to receive
                    attribution, in any reasonable manner requested by
                    the Licensor (including by pseudonym if
                    designated);

                ii. a copyright notice;

               iii. a notice that refers to this Public License;

                iv. a notice that refers to the disclaimer of
                    warranties;

                 v. a URI or hyperlink to the Licensed Material to the
```

```
                    extent reasonably practicable;

            b. indicate if You modified the Licensed Material and
               retain an indication of any previous modifications; and

            c. indicate the Licensed Material is licensed under this
               Public License, and include the text of, or the URI or
               hyperlink to, this Public License.

         For the avoidance of doubt, You do not have permission under
         this Public License to Share Adapted Material.

      2. You may satisfy the conditions in Section 3(a)(1) in any
         reasonable manner based on the medium, means, and context in
         which You Share the Licensed Material. For example, it may be
         reasonable to satisfy the conditions by providing a URI or
         hyperlink to a resource that includes the required
         information.

      3. If requested by the Licensor, You must remove any of the
         information required by Section 3(a)(1)(A) to the extent
         reasonably practicable.


Section 4 -- Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that
apply to Your use of the Licensed Material:

  a. for the avoidance of doubt, Section 2(a)(1) grants You the right
     to extract, reuse, reproduce, and Share all or a substantial
     portion of the contents of the database for NonCommercial purposes
     only and provided You do not Share Adapted Material;

  b. if You include all or a substantial portion of the database
     contents in a database in which You have Sui Generis Database
     Rights, then the database in which You have Sui Generis Database
     Rights (but not its individual contents) is Adapted Material; and

  c. You must comply with the conditions in Section 3(a) if You Share
     all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not
replace Your obligations under this Public License where the Licensed
Rights include other Copyright and Similar Rights.


Section 5 -- Disclaimer of Warranties and Limitation of Liability.

  a. UNLESS OTHERWISE SEPARATELY UNDERTAKEN BY THE LICENSOR, TO THE
```

```
     EXTENT POSSIBLE, THE LICENSOR OFFERS THE LICENSED MATERIAL AS-IS
     AND AS-AVAILABLE, AND MAKES NO REPRESENTATIONS OR WARRANTIES OF
     ANY KIND CONCERNING THE LICENSED MATERIAL, WHETHER EXPRESS,
     IMPLIED, STATUTORY, OR OTHER. THIS INCLUDES, WITHOUT LIMITATION,
     WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR
     PURPOSE, NON-INFRINGEMENT, ABSENCE OF LATENT OR OTHER DEFECTS,
     ACCURACY, OR THE PRESENCE OR ABSENCE OF ERRORS, WHETHER OR NOT
     KNOWN OR DISCOVERABLE. WHERE DISCLAIMERS OF WARRANTIES ARE NOT
     ALLOWED IN FULL OR IN PART, THIS DISCLAIMER MAY NOT APPLY TO YOU.

  b. TO THE EXTENT POSSIBLE, IN NO EVENT WILL THE LICENSOR BE LIABLE
     TO YOU ON ANY LEGAL THEORY (INCLUDING, WITHOUT LIMITATION,
     NEGLIGENCE) OR OTHERWISE FOR ANY DIRECT, SPECIAL, INDIRECT,
     INCIDENTAL, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, OR OTHER LOSSES,
     COSTS, EXPENSES, OR DAMAGES ARISING OUT OF THIS PUBLIC LICENSE OR
     USE OF THE LICENSED MATERIAL, EVEN IF THE LICENSOR HAS BEEN
     ADVISED OF THE POSSIBILITY OF SUCH LOSSES, COSTS, EXPENSES, OR
     DAMAGES. WHERE A LIMITATION OF LIABILITY IS NOT ALLOWED IN FULL OR
     IN PART, THIS LIMITATION MAY NOT APPLY TO YOU.

  c. The disclaimer of warranties and limitation of liability provided
     above shall be interpreted in a manner that, to the extent
     possible, most closely approximates an absolute disclaimer and
     waiver of all liability.


Section 6 -- Term and Termination.

  a. This Public License applies for the term of the Copyright and
     Similar Rights licensed here. However, if You fail to comply with
     this Public License, then Your rights under this Public License
     terminate automatically.

  b. Where Your right to use the Licensed Material has terminated under
     Section 6(a), it reinstates:

       1. automatically as of the date the violation is cured, provided
          it is cured within 30 days of Your discovery of the
          violation; or

       2. upon express reinstatement by the Licensor.

     For the avoidance of doubt, this Section 6(b) does not affect any
     right the Licensor may have to seek remedies for Your violations
     of this Public License.

  c. For the avoidance of doubt, the Licensor may also offer the
     Licensed Material under separate terms or conditions or stop
     distributing the Licensed Material at any time; however, doing so
     will not terminate this Public License.
```

```
  d. Sections 1, 5, 6, 7, and 8 survive termination of this Public
     License.


Section 7 -- Other Terms and Conditions.

  a. The Licensor shall not be bound by any additional or different
     terms or conditions communicated by You unless expressly agreed.

  b. Any arrangements, understandings, or agreements regarding the
     Licensed Material not stated herein are separate from and
     independent of the terms and conditions of this Public License.


Section 8 -- Interpretation.

  a. For the avoidance of doubt, this Public License does not, and
     shall not be interpreted to, reduce, limit, restrict, or impose
     conditions on any use of the Licensed Material that could lawfully
     be made without permission under this Public License.

  b. To the extent possible, if any provision of this Public License is
     deemed unenforceable, it shall be automatically reformed to the
     minimum extent necessary to make it enforceable. If the provision
     cannot be reformed, it shall be severed from this Public License
     without affecting the enforceability of the remaining terms and
     conditions.

  c. No term or condition of this Public License will be waived and no
     failure to comply consented to unless expressly agreed to by the
     Licensor.

  d. Nothing in this Public License constitutes or may be interpreted
     as a limitation upon, or waiver of, any privileges and immunities
     that apply to the Licensor or You, including from the legal
     processes of any jurisdiction or authority.
```