

Cryptnox Shamir — Operating Manual

Release 1.0

Nov 25, 2026

Getting started

1 Overview **2**
- 1.1 Who this is for . . . 2
- 1.2 Why hardware . . . 2
- 1.3 Supported hardware . . . 2
- 1.4 What you get . . . 3
- 1.5 At a glance . . . 3
- 1.6 Next steps . . . 3

2 Concepts **4**
- 2.1 Shamir Secret Sharing (SSS) . . . 4
- 2.2 The two card types . . . 4
- 2.3 The secret (the “seed”) . . . 4
- 2.4 The HSM and the “wrapped seed” . . . 5
- 2.5 The three operations . . . 5
- 2.6 Card states . . . 5

3 Cards **7**
- 3.1 The three card types . . . 7
- 3.2 Card markings . . . 9
- 3.3 Storage card PIN — security levels . . . 9

4 Installation **11**
- 4.1 Requirements . . . 11
- 4.2 Install . . . 11
- 4.3 Verify the installation . . . 11
- 4.4 Global options . . . 12

5 Cheat sheet **13**
- 5.1 I want to run a ceremony . . . 13
- 5.2 I want to reconstruct my secret into a wallet . . . 14
- 5.3 I want to reset a storage card PIN with the Admin card . . . 14
- 5.4 I want to check my backup is intact . . . 15
- 5.5 I want to reset a card so I can reuse it . . . 15

6 The Ceremony **16**
- 6.1 Before you begin . . . 17
- 6.2 Running the ceremony . . . 17
- 6.3 Where the seed goes (the HSM) . . . 19
- 6.4 How it works . . . 20
- 6.5 Card order (share numbers) . . . 20
- 6.6 After the ceremony . . . 21
- 6.7 Resetting a card before a ceremony . . . 22
- 6.8 Importing an existing seed . . . 22

7 Recovery **24**
- 7.1 Before you begin . . . 24
- 7.2 Running recovery . . . 25

© 2026 Cryptnox SA

7.3 How it works . . . 25
7.4 Verifying the result . . . 26
7.5 Operational notes . . . 26

8 Backup Test **27**
8.1 What you need . . . 27
8.2 Running a backup test . . . 27
8.3 How it works . . . 27
8.4 Why a dedicated test mode . . . 27
8.5 The two counters . . . 28
8.6 Interpreting the result . . . 28
8.7 Checking a single card . . . 28

9 Choosing Parameters **30**
9.1 The quorum: `k` of `n` . . . 30
9.2 Seed size . . . 31
9.3 Seed type . . . 31

10 Custody Governance & Checklists **33**
10.1 Roles . . . 33
10.2 Lifecycle at a glance . . . 33
10.3 Cadence (recommended) . . . 34
10.4 Checklist — initial ceremony . . . 34
10.5 Checklist — periodic verification (every 12 months) . . . 34
10.6 Checklist — custodian offboarding (someone leaves) . . . 35
10.7 Checklist — lost / damaged / locked card . . . 35

11 Printable Checklist **36**
11.1 Deployment . . . 36
11.2 Ceremony . . . 36
11.3 Annual verification . . . 36
11.4 Custodian change . . . 37
11.5 Lost / damaged / locked card . . . 37

12 Security Model **38**
12.1 The core guarantee . . . 38
12.2 What each actor can see . . . 38
12.3 What a stolen card reveals . . . 38
12.4 PIN protection . . . 39
12.5 Card sets . . . 39
12.6 Seed wrapping (transport to the HSM) . . . 39
12.7 Audit: the export counter . . . 39
12.8 Assumptions you must uphold . . . 40

13 Command Reference **41**
13.1 Basic usage . . . 41
13.2 Global options . . . 41
13.3 Workflow commands . . . 41
13.4 Reader commands . . . 42
13.5 Orchestrator commands . . . 42
13.6 Storage commands . . . 43
13.7 Pure-software Shamir (no card) . . . 43

© 2026 Cryptnox SA

14 Troubleshooting **45**
14.1 No readers found . . . 45
14.2 Command not allowed (SW=6985) . . . 45
14.3 Wrong data / crypto error (SW=6A80) . . . 45
14.4 Wrong PIN . . . 45
14.5 Resetting a stuck card . . . 46
14.6 Cards still “Loaded” before a new ceremony . . . 46
14.7 Recovery produced the wrong secret . . . 46
14.8 Reconstruction is slow . . . 47

15 License **48**

© 2026 Cryptnox SA

This system lets an organisation split a single master secret — a Bitcoin treasury key, an HSM wrapping key, a recovery seed — into n tamper-resistant smart cards, of which any k are sufficient to reconstruct it. No single card, and no single person, ever holds the secret.

© 2026 Cryptnox SA

1 Overview

The `cryptnox-shamir` system is a hardware-backed scheme for **splitting, distributing, and recovering a single high-value secret** across a set of smart cards, using Shamir's Secret Sharing.

A secret is split into `n` *shares* and written to `n` smart cards. Any `k` of those cards (the *threshold*) can reconstruct the secret; any `k-1` of them reveal **nothing** — not partial information, nothing. This is the mathematical guarantee of Shamir's scheme.

1.1 Who this is for

This documentation targets **institutional custodians** of cryptographic secrets, for example:

- **Corporate Bitcoin treasuries** protecting the private key (or the HSM wrapping key) that controls cold-storage funds.
- **Funds and exchanges** that need quorum-based, auditable control over a master key, with no single point of compromise.
- **Any organisation** subject to a "no single person can move the funds" control requirement, where `k` officers out of `n` must cooperate.

The typical goal is a **disaster-recovery backup** of a key that normally lives inside a Hardware Security Module (HSM): the live key stays in the HSM, and the Shamir cards are the offline, geographically distributed backup that can restore it if the HSM is lost or destroyed.

1.2 Why hardware

A purely software implementation of Shamir's scheme exposes the secret in the memory of the machine that splits or reconstructs it. The Cryptnox system moves the sensitive operations **onto the cards**:

- The seed is generated, split, and reconstructed **inside the orchestrator card**. The host PC never sees it in plaintext.
- Each share is stored **inside a storage card**, encrypted, and protected by a PIN. A stolen card reveals nothing without the PIN, and one share alone is useless.
- Shares move **card-to-card over an encrypted channel**. The host only relays opaque ciphertext.

1.3 Supported hardware

- **Cryptnox Orchestrator card** — performs all cryptography (1 required).
- **Cryptnox Storage cards** — hold one encrypted share each (2 to 12).
- **A contact PC/SC smart card reader.** Contactless / NFC is **not supported** — both applets reject it.

© 2026 Cryptnox SA

Order card sets through the Shamir Secret Sharing Cards page; readers are at shop.cryptnox.com.

1.4 What you get

- A command-line tool, `shamir-cli`, that drives the entire lifecycle: ceremony, backup test, and recovery.
- Guided, interactive workflows that prompt you to insert and swap cards.
- A pure-software Shamir mode for testing the maths without hardware.

1.5 At a glance

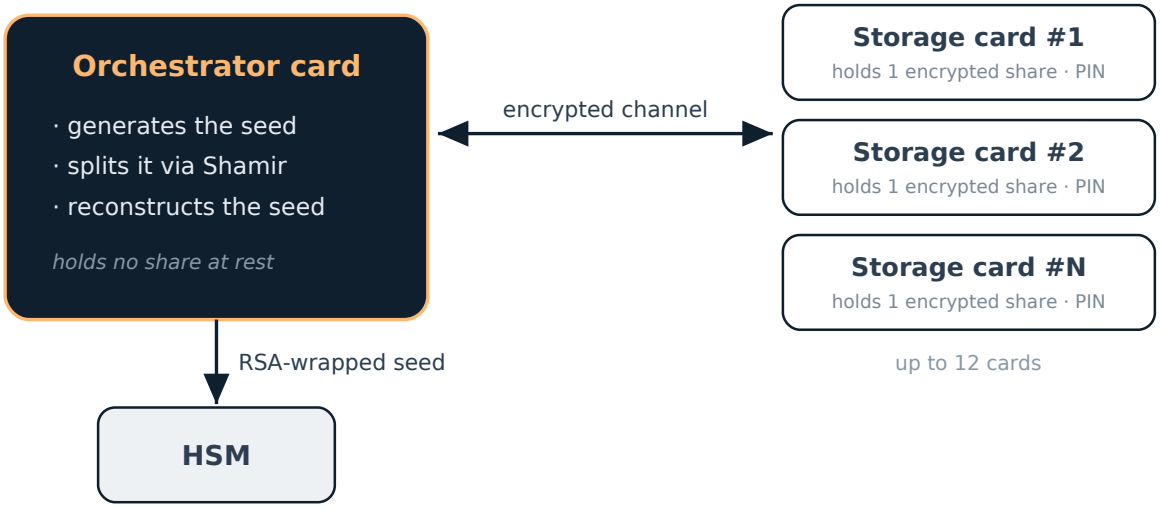

Fig. 1.1: The orchestrator splits the seed across the storage cards over an encrypted channel, and exports the seed RSA-wrapped to the HSM.

1.6 Next steps

- New to the scheme? Read *Concepts*.
- Ready to install? See *Installation*.
- Running your first ceremony? See *The Ceremony*.

See also

- *Security Model* — the threat model, for risk and compliance teams.
- *Choosing Parameters* — choosing the k-of-n quorum for your policy.

© 2026 Cryptnox SA

2 Concepts

This page explains the vocabulary used throughout the rest of the documentation. Read it once before your first ceremony.

2.1 Shamir Secret Sharing (SSS)

A secret is split into `n` **shares**. Any `k` of them (the **threshold**) reconstruct the secret exactly; any `k-1` reveal **nothing** about it.

Notation: a **k-of-n** scheme. For example **2-of-3** means three shares exist and any two of them recover the secret.

The trade-off you choose with `k` and `n`:

- A **larger ``n``** means more redundancy — you can lose more cards.
- A **larger ``k``** means more security — an attacker needs more cards — but also more cards required to recover, so less convenience.

See *Choosing Parameters* for guidance on choosing them.

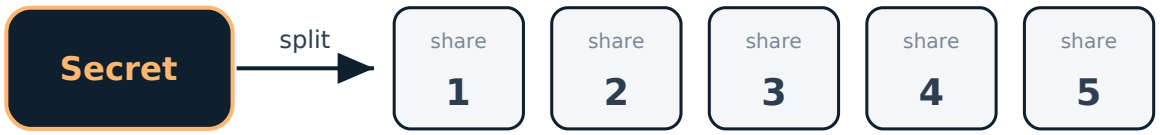

Fig. 2.1: Shamir's Secret Sharing — the secret is split into *n* shares; any *k* of them reconstruct it exactly, while any fewer reveal nothing.

2.2 The two card types

Card	Role	Quantity
Orchestrator	Performs all cryptographic operations: generates the seed, splits it, exports the encrypted shares, and reconstructs the seed during recovery. Holds no share at rest; it is reset and reused between operations.	1
Storage	Securely stores exactly **one** encrypted share, protected by a 4-character PIN. Performs no secret-sharing computation itself.	`n` (2–12)

2.3 The secret (the "seed")

The protected value is called the **seed**. It can be:

- a Bitcoin / secp256k1 private key,

© 2026 Cryptnox SA

- a NIST EC private key (P-256 / P-384 / P-521),
- or any random byte string (e.g. an HSM wrapping key, an AES key).

Sizes range from 16 to 128 bytes. See *Choosing Parameters*.

2.4 The HSM and the "wrapped seed"

In production the live secret is expected to live inside an **HSM** (Hardware Security Module). The Shamir cards are the **offline backup**.

When the orchestrator outputs the seed, it never does so in plaintext. It encrypts ("**wraps**") the seed with the HSM's **RSA public key**. The resulting file — the **wrapped seed** — can only be opened inside the HSM that holds the matching private key. The machine running `shamir-cli` cannot read it.

Note

You supply the HSM public key with `--rsa-key-file`. Only the HSM that holds the matching private key can later open the wrapped seed; the ceremony host never sees the secret in the clear.

2.5 The three operations

Operation	What it does
Ceremony	The one-time event that generates a seed, splits it, and writes the shares to the storage cards. Afterwards the cards are sealed in safes. See *The Ceremony*.
Backup test	A periodic health check that confirms the shares still reconstruct the seed — **without ever exporting the seed**. See *Backup Test*.
Recovery	Gather `k` storage cards and reconstruct the seed (e.g. after the HSM is lost), re-wrapping it for a new HSM. See *Recovery*.

Note

Beyond the full backup test (which needs `k` cards), any **single** storage card can verify **itself** — proving its share is genuine and belongs to the set — without gathering the others. See *Checking a single card*.

2.6 Card states

Each card moves through a small state machine. The most relevant states:

Orchestrator

- `0x00` — Initial (reset). The clean starting point for any operation.
- `0x11`–`0x14` — progressing through a ceremony.
- `0x41`–`0x44` — progressing through a recovery.

© 2026 Cryptnox SA

Storage

- `0x00` — Initial (no share).
- `0x01` — Ready (key generated, awaiting a share).
- `0x02` — Loaded (holds a share).

You can inspect a card's state at any time with `shamir-cli orchestrator info` or `shamir-cli storage info`.

Note

Commands must follow each state machine in order. Sending a command that the current state does not allow returns `SW=6985` ("command not allowed in current state"). When in doubt, reset to `0x00` and start over.

See also

- *Choosing Parameters* — how to choose `k` and `n`.
- *Security Model* — what a stolen card reveals, PINs, audit.
- *The Ceremony* — concepts put into practice.

© 2026 Cryptnox SA

3 Cards

The system uses three types of Cryptnox smart card, distinguished by a colour-coded band and a printed role. All are standard **ID-1** cards (85.6 × 54 mm) with a **gold contact chip**, a unique **serial number (S/N)** printed on the front, and the **CRYPTNOX** / **SHAMIR** branding.

> **Note**
>
> **Contact only.** The Shamir applets reject contactless/NFC — always use a contact reader and insert the card.

3.1 The three card types

3.1.1 Orchestrator (blue)

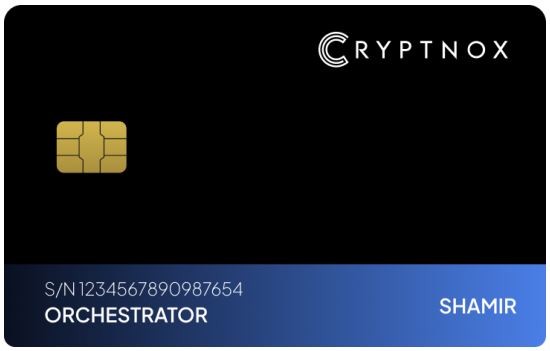

Fig. 3.1: Orchestrator card — performs all cryptography.

Generates the seed, splits it via Shamir, exports the encrypted shares, and reconstructs the seed during recovery. Holds **no** share at rest; it is reset and reused between operations. One per set. See *Concepts*.

On the card: only the **S/N** and branding are factory-set. The back is a printed form — at the ceremony, write the **Bundle** name and an **administrator code** + date in the **Notes** area.

© 2026 Cryptnox SA

3.1.2 Storage (teal)

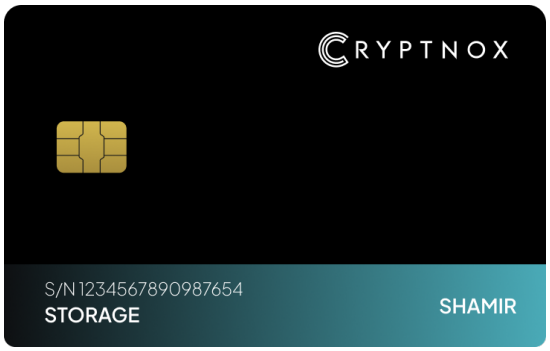

Fig. 3.2: Storage card — holds one encrypted share.

Securely stores exactly **one** encrypted Shamir share, protected by a 4-character PIN. Two to twelve per set. These are the cards you distribute to custodians and store in safes.

On the card: only the **S/N** and branding are factory-set. The back is a printed form you fill in at the ceremony: the **Bundle** name, this card's **share number** (e.g. *#3*), and a **custodian code** + **Ceremony date** in the *Notes* area. **Do not write the total or the threshold on the card** — record the full k-of-n in your custody log only, so a single lost card does not reveal the scheme.

3.1.3 Admin (gold)



Fig. 3.3: Admin card — resets the PIN of a storage card.

Administrative recovery card: it **resets the PIN** of a storage card — for example when a custodian forgets their PIN or leaves the organisation. Keep it under strict control (it governs access recovery to the storage cards).

Important

The Admin card is itself PIN-protected (its own PIN, separate from the storage PINs; default `0000`, change it at provisioning). It will not sign a reset until that PIN is verified, so the card **alone is useless** — an attacker needs both the card **and** its PIN (two-factor). The PIN allows **10** attempts; after that the card **locks permanently** (nothing can reset the Admin card — the fallback is the k-of-n recovery quorum). Store the Admin card and its PIN separately, and never together with the storage cards.

© 2026 Cryptnox SA

On the card: only the **S/N** and branding are factory-set; write the **Bundle** name in the *Notes* area and keep the card with the administrator.

3.2 Card markings

From the factory each card carries only a fixed **S/N** and the branding. At the ceremony you write, in the printed fields on the back, the **Bundle** name and — on storage cards — this card's **share number** (e.g. *#3*), plus a short, **non-identifying** custody code and the ceremony date in the **Notes** area.

Keep the total and threshold off the cards. Record the full k-of-n **only in your custody log**: a single lost or stolen card should not reveal how many cards exist or how many are needed. The card cannot report its share number electronically, so recovery relies on the number you write — write it carefully. See *Card order (share numbers)* and *Choosing Parameters*.

> **Warning**
>
> Write **codes only** in the *Notes* area — **never a full name or a handwritten signature**. Both identify a person tied to the secret and invite **coercion**. The code → name mapping and the holder's signed acknowledgement of receipt live **only** in the secured custody log (see *Card order (share numbers)*).

3.3 Storage card PIN — security levels

When provisioning storage cards, the administrator chooses a PIN policy. Pick the level that matches your custody and recovery requirements.

Level	Policy	Implication
A	Default PIN `0000`	Simplest, **lowest security** — anyone holding the card can read its share. Use only for testing or very low-risk setups.
B	PIN changed (per card)	The share is protected by a secret PIN. **If the PIN is lost, the card (and its share) is unrecoverable** — you must rely on the *k*-of-*n* quorum from the other cards.
C	PIN changed **+ Admin card**	The share is PIN-protected **and** a forgotten/locked PIN can be **reset with the Admin card** — recommended for organisations (resilient to personnel changes and forgotten PINs).

> **Tip**
>
> For an institutional deployment, choose **Level C**: per-card PINs for security, plus an Admin card so a forgotten PIN never forces you to fall back on the quorum. Store the Admin card separately, under its own control.

© 2026 Cryptnox SA

See also

- *Custody Governance & Checklists* — the operational lifecycle and checklists.
- *Security Model* — what a stolen card reveals, PINs, audit.
- *The Ceremony* — provisioning the cards into a set.

© 2026 Cryptnox SA

4 Installation

4.1 Requirements

- **Python 3.10 or newer**
- A **contact PC/SC smart card reader** (for hardware operations)
- On Linux, the PC/SC daemon (`pcscd`) running:

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

4.2 Install

The `shamir-cli` source package is supplied by Cryptnox with your card set. Unpack it and install it from its directory. For hardware operations, include the `card` extra (which pulls in `pyscard`):

```
cd shamir-cli    # the directory supplied by Cryptnox

# With smart card support (physical cards)
pip install -e ".[card]"

# Or, for pure-software Shamir only (no reader)
pip install -e .
```

After installation, the `shamir-cli` command is available on your `PATH`.

> **Note**
>
> This is an *editable* install (`-e`): the command runs the source in `src/shamir_cli`, so any update to the source is picked up immediately. Do not move or delete that directory, or the command will break.

4.3 Verify the installation

```
shamir-cli --version
shamir-cli reader list
```

`reader list` should show your contact reader, for example:

#	Reader Name
0	<your contact reader>

Insert an orchestrator card and confirm it responds:

```
shamir-cli orchestrator info
```

```
Type      O (Orchestrator)
Version   0.9.2
State     0x00 - Initial (reset)
```

4.4 Global options

Every command accepts these global options:

Option	Effect
`-r, --reader TEXT`	Select a reader by partial name match (a substring of its name).
`-S, --simulator`	Use the JCOP simulator on `localhost:8050` instead of hardware.
`--sim-port INT`	Simulator port (default `8050`).
`-v, --verbose`	Print the raw APDU traffic (for debugging).
`-q, --quiet`	Suppress non-essential output.
`--json`	Emit machine-readable JSON (for scripting / integration).

4.4.1 Selecting a reader

When several readers are connected, target one explicitly by a substring of its name (as shown by `reader list`):

```
shamir-cli -r "<reader-name>" orchestrator info
```

The match is a case-insensitive substring of the reader name.

© 2026 Cryptnox SA

5 Cheat sheet

The three things people ask for most, as copy-paste one-liners. Each recipe links to the full procedure — read it before doing this for real. Values like `-n 3 -k 2` and the reader names are **examples**; substitute your own.

> **Note**
>
> **Reader selection.** Pass explicit reader names for a **multi-reader bench** (all cards powered at once). **Omit** them for a **single reader** (you are prompted to swap cards). Find the exact names with `shamir-cli reader list`.

> **Tip**
>
> **See what's on your readers first** — always safe, read-only:
>
> ```
> shamir-cli reader list
> shamir-cli --reader "<admin-reader>" orchestrator info
> shamir-cli --reader "<storage-reader>" storage info
> ```

5.1 I want to run a ceremony

Generate a master secret, split it k-of-n, write the shares to the storage cards, run the backup test, and wrap the seed for the target — here a **Cryptnox wallet** used as the HSM.

```
# 2-of-3, 256-bit random seed, wrapped straight into a Cryptnox wallet
shamir-cli workflow full-ceremony -n 3 -k 2 -s 32 \
    --blessings-bundle ./batch.bundle.json \
    --wallet-hsm "Cryptnox NFC" --wallet-pin 0000 \
    --output-dir ./ceremony-out
```

> **Important**
>
> **A real ceremony consumes the factory blessing bundle** (`*.bundle.json`): it carries the group public key and each card's pre-signed blessing, which the orchestrator needs to accept the storage cards. You may **omit** `--blessings-bundle` only if exactly one `*.bundle.json` sits in the working directory (it is auto-discovered). Only `--test-mode` runs without a bundle.

Wrap for an **external HSM** instead of a wallet with `--rsa-key-file hsm_pubkey.pem` (or `--kms-key-id` for AWS KMS). On a bench, add `--orch-reader ... --storage-readers r1,r2,r3`.

→ Full guide: *The Ceremony*.

© 2026 Cryptnox SA

5.2 I want to reconstruct my secret into a wallet

Collect k storage cards, rebuild the seed on an orchestrator, and import it **directly into a Cryptnox wallet** (the seed never lands in a file).

```
# any 2 of the 3 cards; insert them in share-index order when prompted
shamir-cli workflow full-recovery -k 2 \
    --wallet-hsm "Cryptnox NFC" --wallet-pin 0000 \
    --pin 0000
```

Re-wrap for a **new HSM** instead with `--rsa-key-file new_hsm_pubkey.pem --output-file recovered_seed.bin`.

→ Full guide: *Recovery*.

5.3 I want to reset a storage card PIN with the Admin card

Reset a forgotten/locked PIN **without knowing the old one**, using the Admin card (nonce challenge, signed by the Admin, verified by the storage card).

```
# Prerequisite (set-once, FRESH card only — normally done at the factory):
# the storage must carry the admin public key. On a fresh (0x00) card:
shamir-cli --reader "<admin-reader>" admin get-pubkey          # -> <admin_pub hex>
shamir-cli --reader "<storage-reader>" storage set-admin-pubkey <admin_pub hex>

# Reset the PIN — bench (two readers, nothing to swap):
shamir-cli workflow reset-pin --new-pin 1234 --admin-pin 0000 \
    --storage-reader "<storage-reader>" --admin-reader "<admin-reader>"

# ...or single reader (insert/swap the cards when prompted):
shamir-cli workflow reset-pin --new-pin 1234 --admin-pin 0000
```

Verify afterwards with `shamir-cli --reader "<storage-reader>" storage verify-pin 1234`.

Important

The Admin card has its own PIN (`--admin-pin`, default `0000`): it refuses to sign a reset until that PIN is verified — so a lost/stolen Admin card **alone** cannot reset anything (two-factor: possession + PIN). It allows **10** attempts and its lockout is **irreversible** (nothing sits above the Admin card), so keep the admin PIN safe and change it from the default at provisioning (`shamir-cli admin change-pin <new> --pin <current>`).

Note

The admin public key must be provisioned on the storage card **before** the ceremony loads its share (it is accepted only on a fresh card). The factory does this during personalization; a card already carrying a share cannot be made admin-resettable retroactively.

© 2026 Cryptnox SA

> **Note**
>
> If you **know** the current PIN and just want to change it (no Admin card), use `storage verify-pin <old>` then `storage change-pin <new>`.

→ Context and custody policy: *Custody Governance & Checklists*.

5.4 I want to check my backup is intact

Confirm `k` storage cards still reconstruct the original secret — **without exporting the seed and without incrementing the read counter** of the cards. This is the safe, repeatable health check.

```
shamir-cli workflow backup-test -k 2 --pin 0000
```

It reports the reconstructed checksum for you to compare against the value recorded at the ceremony. Read counters stay untouched, so you can run it as often as you like.

→ Full guide: *Backup Test*.

5.5 I want to reset a card so I can reuse it

Return a card to the factory state (state `0x00`) to start a fresh ceremony or re-provision it.

```
# Orchestrator — back to 0x00 (e.g. after a recovery left it at 0x44):
shamir-cli --reader "<admin-reader>" orchestrator reset

# Storage — requires the current PIN AND a typed confirmation:
shamir-cli --reader "<storage-reader>" storage reset --pin 0000
#   -> prompts: Type 'I understand' to proceed

# ...in a script, skip the confirmation with --yes:
shamir-cli --reader "<storage-reader>" storage reset --pin 0000 --yes
```

> **Warning**
>
> Resetting a **storage** card **permanently destroys its share** and cannot be undone. The command makes you type `I understand` first; `--yes` bypasses that for automation. Only reset a card whose share you no longer need, or after a successful recovery. Resetting the **orchestrator** is harmless — it holds no long-term secret, so it needs no confirmation.
>
> The **Admin card has no reset**: its key is the fleet's PIN-reset root (no rotation path), so it is permanent — decommissioning an Admin card is a factory re-flash, not a field operation.

→ Card states and lifecycle: *Concepts* / *Custody Governance & Checklists*.

© 2026 Cryptnox SA

6 The Ceremony

The **ceremony** is the one-time event in which you generate a master secret, split it into shares, and write each share to a storage card. It is the most important procedure in the system and should be performed deliberately, under controlled conditions.

> **Note**
>
> **Generate vs import.** In `full-ceremony` (this page) the seed is **generated on the orchestrator** — you never enter it, and it never exists off the card. This is the most secure option and the recommended default. To back up a key that **already exists** (for example a treasury key you hold elsewhere), use the **import ceremony** instead — see *Importing an existing seed*.

Throughout this page we use a **2-of-3** ceremony (three cards, any two recover) with a 256-bit seed **purely as a worked example**. It is not a fixed or recommended value — substitute your own `-n`, `-k` and `-s` for any supported configuration (*Choosing Parameters*).

> **Note**
>
> **2-of-3 is just an example.** The system supports `n` from 2 to 12 and `k` from 2 to 7. Choose the quorum that matches your custody policy.

Important

The provisioned count is a ceiling, not a quota. A set ships with a fixed number of storage cards — that is the **maximum** `n`. At the ceremony you choose **any** valid `k`-of-`n` with `n` $\leq$ the cards you hold: you may use fewer and leave the spares unused. You only cannot set `n` *higher* than the number of cards. The k-of-n is **not** marked at the factory — you write it on the cards (see *Card order (share numbers)*), so you are free to decide it right up to the ceremony.



Fig. 6.1: The ceremony at a glance — the seed is generated and split on the orchestrator, distributed to the storage cards, verified by a backup test, then exported RSA-wrapped to the HSM.

6.1 Before you begin

Prepare:

- **1 orchestrator card** in state `0x00` (reset if needed — see below).
- **``n`` storage cards** from the same set, in state `0x00` (3 for this example).
- The **RSA public key of your HSM**, exported to a file (e.g. `hsm_pubkey.pem`). This is the key the seed will be wrapped for.
- The **factory blessing bundle** for this set (`*.bundle.json`). It lets the orchestrator accept the storage cards of this set. It is produced when the cards are enrolled at the factory. Only `--test-mode` runs without one.

Production checklist

- Perform the ceremony on a **trusted, offline machine** in a controlled location, ideally with multiple officers present (four-eyes principle).
- Use your real **HSM RSA public key** via `--rsa-key-file`. Without it, the tool generates a throwaway test key and the resulting wrapped seed is **not usable** in production.
- **Write each card's share number as you load it** (#1, #2, #3 ...) and keep the cards in that order. The card cannot report its own share index, so recovery relies on what you write on the back. See *Card order (share numbers)*.

6.2 Running the ceremony

```
shamir-cli workflow full-ceremony \
  -n 3 \
  -k 2 \
  -s 32 \
```

```
 -t f \
 --blessings-bundle ./batch.bundle.json \
 --rsa-key-file hsm_pubkey.pem \
 --output-dir ./ceremony-output
```

Note

`--blessings-bundle` may be **omitted** only if exactly one `*.bundle.json` sits in the working directory — it is then auto-discovered. A real ceremony always consumes one; only `--test-mode` runs without a bundle.

Options:

Option	Meaning
`-n, --total`	Total number of storage cards / shares (2–12).
`-k, --threshold`	Minimum cards required to recover (2–7, `k` $\leq$ `n`).
`-s, --seed-size`	Seed size in bytes: `16` | `32` | `48` | `64` | `66` | `128`.
`-t, --seed-type`	`f` random · `r` EC r1 key · `k` secp256k1 key.
`--blessings-bundle`	Factory blessing bundle for this card set (`*.bundle.json`). Auto-discovered in the working dir if omitted.
`--wallet-hsm`	Use a Cryptnox wallet on this reader as the HSM: the seed is imported straight into it, nothing on disk. The wallet must be initialised and hold no seed (`NO_SEED`). See *Where the seed goes (the HSM)*.
`--wallet-pin`	PIN of the wallet used with `--wallet-hsm` (default `0000`).
`--rsa-key-file`	HSM RSA public key (PEM/DER) used to wrap the seed.
`--kms-key-id`	AWS KMS asymmetric RSA key used to wrap the seed. Pass the **full ARN** of an existing key. See *Where the seed goes (the HSM)*.
`--kms-region`	AWS region for `--kms-key-id` (default `eu-north-1`).
`--output-dir`	Where to write the wrapped seed file.

With a single reader, the tool prompts you to insert and swap cards:

```
Insert Orchestrator card and press Enter...
Insert Storage card #1 and press Enter...
Swap back to Orchestrator card and press Enter...
Insert Storage card #2 and press Enter...
...
```

© 2026 Cryptnox SA

Tip

With **two readers** you can keep the orchestrator in one and only change the storage card in the other, which reduces card swaps considerably.

6.3 Where the seed goes (the HSM)

The generated seed never touches the disk in the clear — it is RSA-wrapped inside the orchestrator and handed to **exactly one** recipient (the "HSM"). Choose one of:

Recipient	How
Cryptnox wallet	`--wallet-hsm <reader>` (+ `--wallet-pin`). The seed is imported straight into the wallet on-card; nothing is written to disk. The wallet must be initialised and `NO_SEED`. Both wallet generations are accepted (the SDK reports either `Basic` or `BasicG1`).
AWS KMS	`--kms-key-id <arn>` (+ `--kms-region`). The seed is wrapped to an AWS KMS asymmetric RSA key; KMS holds the private half and can decrypt it later. The wrapped blob is written to `--output-dir`.
External RSA HSM	`--rsa-key-file <pem>`. The seed is wrapped to that public key and written to `--output-dir` as `wrapped_seed.bin`.

6.3.1 AWS KMS — credentials and key

The KMS recipient talks to AWS through **boto3**. No secret is stored in this tool; credentials come from the standard AWS chain:

1. Environment variables — `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_DEFAULT_REGION`; or
2. a shared profile written by `aws configure` (`~/.aws/credentials` + `~/.aws/config`); or
3. an IAM role (when running on an EC2 instance / container).

Create an access key in the AWS console under **IAM** → **Users** → **<user>** → **Security credentials** → **Access keys** → **Create access key** (use case *Command Line Interface*). The **Secret Access Key is shown only once** — save it. Then:

```
aws configure          # Access Key ID + Secret + region (e.g. eu-north-1)
aws sts get-caller-identity   # verify: prints your ARN, not an error
```

Important

Pass the **full key ARN** to `--kms-key-id`, e.g. `arn:aws:kms:eu-north-1:<account>:key/<key-id>`. A **bare key-id** (just the UUID) is interpreted as an *alias name* and the tool will try to **create** a key — which fails with `AccessDeniedException` on `kms:CreateKey` unless you hold that permission. An `alias/<name>` is also accepted and is created on first use (needs `kms:CreateKey` + `kms:CreateAlias`).

The key must be an **asymmetric RSA** key with usage *ENCRYPT_DECRYPT* (RSA-2048 or larger). Minimum permissions to use an existing key: `kms:GetPublicKey` (to wrap) and

`kms:Decrypt` (to recover / verify).

On success the ceremony wraps the seed to KMS, writes `kms_wrapped_seed.bin`, and re-decrypts it through KMS to confirm the checksum matches the on-card seed (`KMS decrypt check:  checksum match = True`).

6.4 How it works

1. **Reset the orchestrator** to a clean state and enter Ceremony mode.
2. **Register each storage card.** Each card and the orchestrator authenticate one another, and the orchestrator records the card.
3. **Load the HSM RSA wrapping key.**
4. **Generate the seed** inside the orchestrator. It returns a *checksum* (a short fingerprint) — **write it down**.
5. **Export each share**, encrypted for its specific storage card, and load it into that card.
6. **Backup test.** The orchestrator reads the shares back and confirms they reconstruct the original seed. (All `n` shares are exercised.)
7. **Export the RSA-wrapped seed** to `ceremony-output/wrapped_seed.bin`.

A successful run ends with a summary panel:

```
──────────────────────── Summary ────────────────────────
Ceremony Complete
  Shares: 3 total, 2 required
  Seed: 256-bit, type=f
  Seed checksum: 33 F4 A2 59 31 57 9D 19
  Wrapped seed: wrapped_seed.bin
─────────────────────────────────────────────────────────
```

6.5 Card order (share numbers)

The card backs are a **printed form** — the factory does not mark the k-of-n. As you run the ceremony, **you write** on each storage card's back:

- the **bundle name** (the set identifier),
- this card's **share number** — card #1 first, #2 second, and so on, written as you load it, so each label matches the share it actually stores.

Important

Keep the total and threshold off the cards. Write only the share number; record the full k-of-n in your **custody log**. A single lost or stolen card should not reveal how many cards exist or how many are needed.

The card does not report its share number electronically, so recovery relies on the number **you wrote**: at recovery, insert the cards in the same order. Inserting them out of order

reconstructs the **wrong** secret (the checksum will not match). Write legibly and double-check.

6.5.1 Annotate the back of each card

The back of each card is a printed form with empty fields. In addition to the set parameters above, fill in a short, **non-identifying** custody marker in the **Notes** area (see *Card markings*):

- **Orchestrator card** — the **Bundle** name, the **administrator's operator code**, and the **ceremony date**.
- **Each storage card** — the **custodian code** (assigned by the administrator; the scheme is up to your organisation, e.g. `C-03`) and the **ceremony date**.

Warning

Keep the cards **anonymous**: put **no full name and no handwritten signature on the card** — both identify a person and make them a target for **coercion**. The acknowledgement of receipt (the holder's signature against their code) and the **code → name** mapping live **only in your custody log**, stored securely.

This keeps each card's role readable at a glance while keeping the people involved untargetable from the card alone; accountability lives in the log.

6.6 After the ceremony

You now hold:

- **``n`` storage cards**, each in state `0x02` (loaded) and labelled. Store them in **separate, secure locations** (different safes / sites).
- **One** `wrapped_seed.bin` **file** — import it into your HSM using your standard RSA key-import procedure. After import, **destroy the file**; it is only an encrypted transport artifact.
- **One orchestrator card**, reset and reusable.

Tip

Before sealing each card in its safe, you can spot-check it on its own with `shamir-cli storage pedersen-verify` — it confirms that one card holds a valid share for this set, without needing the others. See *Checking a single card*.

Record, in your custody log: the **seed checksum**, the `k` / `n` parameters, the seed size/type, and the physical location assigned to each labelled card.

Warning

The seed is **never** exported in plaintext. The orchestrator destroys its in-memory copy on reset and after export. The only way to recover the seed later is the wrapped seed (via the

HSM private key) **or** a *Recovery* from `k` storage cards.

6.7 Resetting a card before a ceremony

If a card is not in state `0x00`:

```
# Orchestrator
shamir-cli orchestrator reset

# Storage (PIN required)
shamir-cli storage reset --pin 0000
```

See *Troubleshooting* if a reset is refused.

6.8 Importing an existing seed

Use **import** when the secret you want to protect **already exists** (for example a treasury key generated elsewhere) — rather than letting the orchestrator generate a fresh one. It is the very same `full-ceremony` command with one extra flag: `--secret-file` (or `--secret`).

You **must** supply the secret to import, either as a file (recommended) or as a hex string:

```
shamir-cli workflow full-ceremony \
  -n 3 \
  -k 2 \
  -s 32 \
  --orch-reader "<orchestrator reader>" \
  --storage-readers "<reader1>,<reader2>,<reader3>" \
  --blessings-bundle ./cards.bundle.json \
  --secret-file ./treasury.hex
```

- `--secret-file <file>` — a file holding the secret. It may be **hex text** (e.g. `00112233...`) or **raw bytes**; either way its length must equal `-s`.
- `--secret <hex>` — the secret inline as hex. Prefer `--secret-file` so the secret never lands in your shell history.

Everything else is identical to a normal ceremony — the same real cards, the same factory bundle, the same on-card VSS split and per-card verification — **except**:

- the seed is **imported** (RSA-encrypted to the orchestrator) instead of generated on-card; and
- it is **not** re-wrapped to an HSM (you already hold it), so the HSM options (`--wallet-hsm` / `--rsa-key-file` / `--kms-key-id`) do not apply.

Note

`workflow ceremony-import` is a separate, **software-only** command (virtual storage cards, throwaway test material) kept for simulator demos. For real enrolled cards, use `full-ceremony --secret-file` as above.

© 2026 Cryptnox SA

Important

The secret is **required**. `ceremony-import` exists precisely to protect a key you already hold — it never invents a random one. If you want a fresh random seed, use `full-ceremony` instead: it generates it **on-card** with the smart card's hardware TRNG, which is stronger than any host-side generator.

Note

The secret is imported in **encrypted** form (wrapped to the orchestrator's RSA key); it is never sent to the card in the clear, and the orchestrator never exposes it. The secret is not printed to the console — only its length and the resulting checksum are shown.

See also

- *Choosing Parameters* — choosing `n`, `k`, seed size and type.
- *Backup Test* — confirm the backup is healthy afterwards.
- *Recovery* — restore the seed from `k` cards.
- *Security Model* — what a stolen card reveals, PINs, audit.

© 2026 Cryptnox SA

7 Recovery

Recovery reconstructs the seed from `k` storage cards — for example after the production HSM is lost or destroyed — and re-wraps it for a new HSM.

This page uses a **2-of-3** deployment (any two of the three cards) **purely as a worked example** — the procedure is identical for any supported threshold (`k` from 2 to 7, `n` from 2 to 12).

> **Note**
>
> **2-of-3 is just an example.** Recover with whatever `k` your deployment was created with.



Fig. 7.1: Recovery at a glance — *k* storage cards are collected, the seed is reconstructed on the orchestrator, then re-wrapped for the new HSM.

7.1 Before you begin

Prepare:

- **1 orchestrator card** (any orchestrator in state `0x00`; it need not be the one used for the ceremony).
- **``k`` storage cards** holding valid shares, with their PINs known.
- The **RSA public key of the new HSM** where the seed will be restored (e.g. `new_hsm_pubkey.pem`).
- Your record of **which physical card holds which share index** (see *Card order (share numbers)*).

Note

Card order. On current firmware each storage card reports its **own** share index, so recovery reconstructs correctly **whatever order you insert the cards** in — the tool reads the true index from each card. Insertion order only matters on **older cards** that do not expose their index, where the tool falls back to the order of insertion (first inserted = share #1, ...) and a wrong order silently yields the **wrong secret**. Either way, **verify the checksum** (below) before trusting the result — it is the definitive check.

7.2 Running recovery

```
shamir-cli workflow full-recovery \
  -k 2 \
  --rsa-key-file new_hsm_pubkey.pem \
  --output-file recovered_seed.bin \
  --pin 0000
```

Options:

Option	Meaning
`-k, --threshold`	Number of shares to collect (must equal the ceremony's `k`).
`--rsa-key-file`	RSA public key of the **new** HSM to wrap the recovered seed for.
`--output-file`	Where to write the wrapped recovered seed.
`--pin`	Storage card PIN (default `0000`).

The tool prompts for the orchestrator and then each storage card:

```
Insert Orchestrator card and press Enter...
Insert Storage card #1 and press Enter...
Swap back to Orchestrator card and press Enter...
Insert Storage card #2 and press Enter...
Swap back to Orchestrator card and press Enter...
Recovery checksum: 33 F4 A2 59 31 57 9D 19
Wrapped seed saved to recovered_seed.bin
```

7.3 How it works

1. The orchestrator is reset and enters **Recovery** mode.
2. The new HSM RSA wrapping key is loaded; the orchestrator generates an ephemeral encryption key.
3. For each of the `k` cards:
   - You verify the PIN.
   - The storage card **re-encrypts** its share to the orchestrator's key and hands it over. The host only relays opaque ciphertext.
   - The orchestrator decrypts the share internally.

4. The orchestrator reconstructs the seed (Lagrange interpolation, performed on-card) and returns a **checksum**.
5. The orchestrator exports the **RSA-wrapped** recovered seed to the output file.

7.4 Verifying the result

Always compare the `Recovery checksum` against the seed checksum you recorded at the ceremony.

- **Match** → the correct seed was reconstructed. Import `recovered_seed.bin` into the new HSM, then destroy the file.
- **Mismatch** → you collected the wrong cards, or inserted them in the wrong order. Do **not** import the result. Re-run, inserting the correctly labelled cards in share-index order.

7.5 Operational notes

- **Export counter.** Every real share read during a recovery increments a counter inside the storage card. Inspect it for audit with `shamir-cli storage read-counter`. An unexpected increase means a share was read outside a sanctioned recovery.
- **PIN lockout.** Three consecutive wrong PINs lock the card. Guard the PINs and the lockout budget.
- **The PC never sees a plaintext share.** Shares move card-to-card over an encrypted channel; the host only relays ciphertext.

Note

Recovery only *reads* the storage cards — it does **not** consume or alter the shares. You can recover as many times as needed (each read increments the audit counter).

See also

- *The Ceremony* — how the shares and labels were created.
- *Backup Test* — verify shares **without** exporting the seed.
- *Wrong PIN* — recovering from PIN errors.
- *Security Model* — what the host can and cannot see.

© 2026 Cryptnox SA

8 Backup Test

A **backup test** confirms that your shares still reconstruct the seed — **without ever exporting the seed**. It is the safest routine to run on a schedule (for example annually, or whenever cards are moved between sites) to prove that your custody backup is still healthy.

> **Note**
>
> This is the recommended periodic health check. It needs no HSM key, does **not** export the secret, and does **not** consume the storage cards' read budget.

8.1 What you need

- **1 orchestrator card** in state `0x00`.
- **``k`` storage cards** with valid shares.

8.2 Running a backup test

```
shamir-cli workflow backup-test -k 2 --pin 0000
```

`-k, --threshold`
: Number of shares to test (must equal the deployment's threshold).

`--pin`
: Storage card PIN (default `0000`).

8.3 How it works

1. The orchestrator is reset and enters **Test** mode (`0x30`).
2. For each of the `k` cards, the share is read back in a **test-only read** that bumps the card's dedicated **test counter** — never the export (recovery) counter (see *The two counters*).
3. The orchestrator reconstructs the seed internally and computes its checksum.
4. It reports **PASS** or **FAIL**. The seed is **never** exported.

8.4 Why a dedicated test mode

Compared to a recovery, the backup test is deliberately safer for routine use:

- **No PIN spend risk on the audit counter** — the test read does **not** increment the recovery export counter.
- **The seed cannot leave the card** — Test mode has no export capability, so even a compromised host cannot exfiltrate the secret during a check.

- **Non-destructive** — it only reads; shares are untouched.

8.5 The two counters

Every storage card keeps **two independent, monotonic counters** (each only ever goes up, never resets except on a full card reset):

Counter	Meaning
Export counter	Bumped on every **real recovery** read, when the share is re-encrypted out for reconstruction and the seed is exported. This is the audit metric for genuine exports — a backup test **never** moves it.
Test counter	Bumped on every **backup test** read. The share *is* re-encrypted during a test, so it is worth recording; the test counter leaves an append-only trace of tests **without polluting the export counter**.

Read either at any time:

```
shamir-cli storage read-counter        # export counter (real recoveries)
shamir-cli storage read-test-counter   # backup-test counter
```

So you can run a backup test as often as you like: the **export counter stays put**, while the **test counter** grows — a clean separation between "this card has been recovered N times" and "this card has been test-verified M times".

8.6 Interpreting the result

- **PASS** — the tested shares reconstruct correctly. Your backup is healthy.
- **FAIL** — the shares did not reconstruct. Investigate immediately: a card may be damaged, or you may have collected cards from different ceremonies.

> **Tip**
>
> Rotate which `k` cards you test over successive checks, so that every card is exercised over time — not just the same `k` each year.

8.7 Checking a single card

The backup test above reconstructs the seed and therefore needs `k` cards together. Sometimes you only want to confirm that **one** card is healthy — for example when a single card comes back from a remote site, or before you re-seal it in its safe. For that, a storage card can verify **itself**:

```
shamir-cli storage pedersen-verify
```

The card proves, on its own, that the share it holds is a genuine part of this set's split — **without reconstructing the seed** and **without gathering the other cards**. It reports the card's **share number** and a **set fingerprint** (shown as `PID`) that identifies which ceremony the card belongs to.

© 2026 Cryptnox SA

Note

A single card is enough: this is a per-card cryptographic check, not a reconstruction. It exports nothing, needs no HSM key, and does **not** move either counter.

8.7.1 Checking a whole set at once

With one card per reader you can verify a whole set in a single command, and confirm the cards are mutually consistent:

```
shamir-cli storage pedersen-verify --readers "<reader-1>,<reader-2>,<reader-3>"
```

The result is a per-card table followed by an overall verdict:

Verdict	Meaning
BATCH OK	Every card verified, all report the **same** set fingerprint, and each carries a **distinct** share number. The set is coherent.
MISMATCH	A card reports a **different** fingerprint — it belongs to another ceremony. Check you have not mixed cards from different sets.
FAILED	A card failed its check, or **two cards carry the same share number** (a possible copy). Investigate immediately.

Tip

Run a single-card check whenever a card changes hands or location — it is quick, needs only that one card, and surfaces a mixed-up or duplicated card long before a recovery would.

See also

- *Recovery* — the full restore (exports the seed to a new HSM).
- *The Ceremony* — where the shares were created.

9 Choosing Parameters

Before a ceremony you choose three things: the **quorum** (`k` of `n`), the **seed size**, and the **seed type**. This page helps you pick values that match your custody policy.

9.1 The quorum: `k` of `n`

- **``n``** is the total number of cards (and shares).
- **``k``** is how many of them are required to reconstruct the secret.

The intuition:

- A larger **``n``** gives more **redundancy** — you can afford to lose more cards (you can lose up to `n - k` and still recover).
- A larger **``k``** gives more **security** — an attacker must compromise more cards — at the cost of needing more people/locations to recover.

> **Note**
>
> The two are independent. `3-of-5` survives the loss of 2 cards **and** requires 3 to recover; `2-of-5` survives 3 losses but only needs 2 to recover (weaker). Pick `k` from your security policy, then `n` from how much redundancy you want on top.

Typical configurations:

Use case	`n`	`k`	Notes
Personal / small treasury	3	2	Three sites, any two recover. Survives one loss.
Small team / four-eyes	5	3	Majority required; survives two losses.
Enterprise	7	4	Robust to multiple losses and absentees.
Maximum distribution	12	7	The system's hard limits.

Constraints: $n \in [2, 12]$, $k \in [2, 7]$, and $k \leq n$.

> **Note**
>
> `n` is also capped by the **number of storage cards you hold**: a set's card count is the maximum `n`. You may use fewer (leaving spares unused), but never more. The k-of-n is your choice at the ceremony — it is not marked at the factory. See *Card order (share numbers)*.

© 2026 Cryptnox SA

Warning

`k = 1` is **not** supported (and would be insecure — a single card would reveal the secret). The minimum threshold is `2`.

9.2 Seed size

The seed size is given in **bytes** with `-s`. Choose it to match the secret you are protecting.

Size	Bits	Typical use
16	128	A 128-bit AES key; BIP-39 minimum entropy.
32	256	**Most common.** secp256k1 / NIST P-256 private key, BIP-32/39 seed material, 256-bit symmetric key.
48	384	NIST P-384 private key.
64	512	BIP-39 maximum, HMAC-SHA-512 key.
66	521	NIST P-521 private key.
128	1024	Very large secrets.

Note

For a Bitcoin treasury key, the common choice is `-s 32` (256-bit). For an HSM AES wrapping key, choose the size of that key (`16` or `32`).

9.3 Seed type

The seed type (`-t`) tells the orchestrator how to generate or interpret the secret:

`-t f`

: **Fully random** bytes with no structure. The most flexible choice — use it for AES keys, HMAC keys, or any opaque secret. Valid sizes: `16, 32, 48, 64, 128`.

`-t r`

: A **NIST EC "r1" private key** (a valid scalar for P-256 / P-384 / P-521). Use when the secret must be a valid EC private key on those curves. Valid sizes: `32, 48, 66`.

`-t k`

: A **secp256k1 private key** — the Bitcoin / Ethereum curve. Valid size: `32`.

Tip

If you are protecting a **Bitcoin** key, use `-t k -s 32`. If you just need a random secret (e.g. an HSM wrapping key), use `-t f`.

See also

- *The Ceremony* — where these parameters are supplied.

- *Security Model* — how the quorum maps to your threat model.

© 2026 Cryptnox SA

10 Custody Governance & Checklists

This page turns the system into an **operating procedure** for an organisation: who does what, and the recurring controls that keep the backup healthy over its lifetime. The checklists are written to be printed and signed (see *Printable Checklist* for a one-page version).

10.1 Roles

- **Custody administrator** — runs the ceremony, holds the Admin card, owns this procedure and the custody log.
- **Custodians** — each holds one storage card (and its PIN), in a separate secure location.
- **Quorum** — any k custodians can, together, reconstruct the secret.

10.2 Lifecycle at a glance

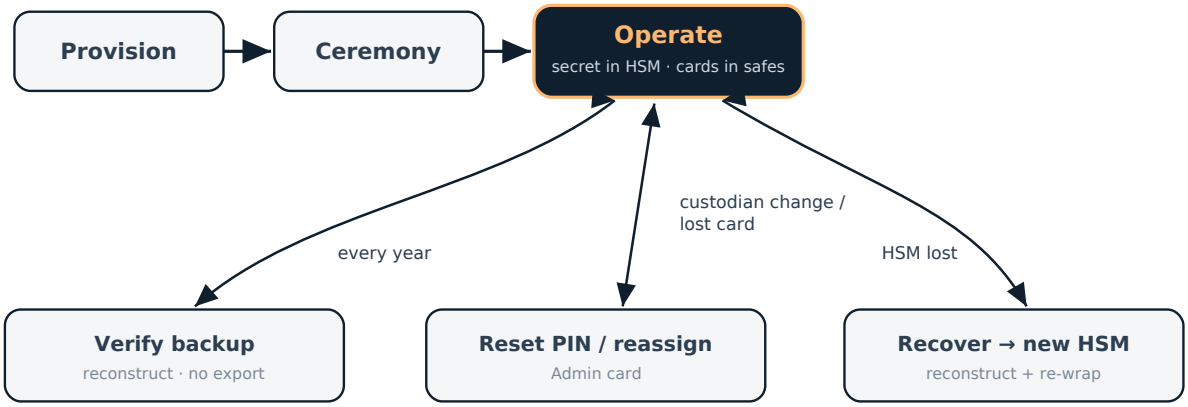

Fig. 10.1: The custody lifecycle: once **Operate** is reached, the recurring events (annual verification, custodian/card changes, recovery) each return to **Operate**.

1. **Provision** the cards into a matched set (factory step).
2. **Ceremony** — generate and split the secret; distribute the storage cards. (*The Ceremony*)
3. **Operate** — the live secret is in the HSM; the cards are the offline backup, sealed in safes.
4. **Verify periodically** — confirm the shares still reconstruct, without exposing the secret. (*Backup Test*)
5. **Handle changes** — personnel changes, lost/damaged cards.
6. **Recover** if ever needed. (*Recovery*)

© 2026 Cryptnox SA

10.3 Cadence (recommended)

When	Action
At setup	Ceremony + record the seed checksum and the custody log.
Every 12 months	Backup test (verify the shares reconstruct). A test needs a quorum of `k` cards, so it only exercises the cards used. **If you can gather all ``n`` cards, test them all** (every card verified). Otherwise test `k` and **rotate** which ones each year, so every card is eventually exercised. (With `n = k` there is nothing to rotate.)
Whenever a custodian leaves	Recover their card, reset its PIN (Admin card) and reassign — or re-ceremony if your policy requires fresh shares.
On any card loss/damage	Assess quorum impact; re-ceremony if you risk dropping below `k`.
After any recovery	Check the export counters; review who accessed what.

10.4 Checklist — initial ceremony

```
[ ] Cards provisioned as a matched set; PIN level chosen (A/B/C)
[ ] Trusted, offline machine; multiple officers present (four-eyes)
[ ] Real HSM RSA public key provided (--rsa-key-file), not a test key
[ ] Factory blessing bundle for this set present (*.bundle.json)
[ ] Ran: shamir-cli workflow full-ceremony -n N -k K -s SIZE \
        --blessings-bundle BUNDLE.json --rsa-key-file ...
[ ] Each storage card's share number written as it is loaded (#1, #2, ...)
[ ] Card backs filled in (codes only, never full names, no signature):
    all = Bundle + code + ceremony date (Notes); storage = also share number
[ ] Total (n) and threshold (k) in the custody log ONLY, never on the cards
    (full names and signatures also live only in the log)
[ ] Backup test passed at the end of the ceremony
[ ] Wrapped seed imported into the HSM, then the file destroyed
[ ] Custody log recorded: seed checksum, k/n, seed size/type, per-card holder
[ ] Storage cards sealed in separate secure locations
[ ] Orchestrator card reset; Admin card stored separately under its own control
```

10.5 Checklist — periodic verification (every 12 months)

```
[ ] Collect the cards: all n if you can gather them, else k (rotate which
    k each year so every card is eventually tested)
[ ] Ran: shamir-cli workflow backup-test -k K
[ ] RESULT = PASS  (seed reconstructs; it is NEVER exported in this mode)
[ ] Read each card's export counter; matches the custody log
[ ] Cards returned to their secure locations; log updated & signed
```

10.6 Checklist — custodian offboarding (someone leaves)

```
[ ] Retrieve the departing custodian's storage card
[ ] Reset its PIN with the Admin card (level C) and set a new PIN, OR
    re-provision the card if its PIN is unknown/locked
[ ] Reassign the card to the new custodian; update the custody log
[ ] Run a backup test to confirm the set still reconstructs
[ ] If policy requires fresh shares, run a new ceremony and retire the old set
```

10.7 Checklist — lost / damaged / locked card

```
[ ] Note which share index is affected; update the custody log
[ ] Forgotten/locked PIN, card in hand -> reset PIN with the Admin card (level C)
[ ] Card lost or destroyed -> count remaining usable cards
    [ ] still >= k usable  -> recovery is unaffected; plan a re-ceremony to
        restore full redundancy
    [ ] < k usable         -> the secret can no longer be reconstructed from
        the cards; recover from the HSM / re-establish the backup immediately
```

See also

- *Cards* — the cards and the PIN security levels (A/B/C).
- *Backup Test* — the periodic health check.
- *Recovery* — restoring the seed from `k` cards.
- *Printable Checklist* — one-page printable version.

© 2026 Cryptnox SA

11 Printable Checklist

A condensed, sign-off version of the custody controls. Print this page and keep the signed copies with your custody log. Full detail is in the Governance section of the documentation.

> **Tip**
>
> To print, open this page in a browser and use **Print** → **Save as PDF**.

11.1 Deployment

```
Set ID: _____________   Date: ___________   k-of-n: ____ / ____   Seed: ____ B
Seed checksum: ________________________________   PIN level (A/B/C): ____
```

11.2 Ceremony

```
[ ] Cards from one set; PIN level chosen
[ ] Offline trusted machine; four-eyes
[ ] Real HSM RSA public key used (--rsa-key-file)
[ ] Ceremony completed; backup test PASSED
[ ] Each storage card's share number written as it is loaded (#1, #2, ...)
[ ] Card backs filled in, codes only (never full names, no signature):
    all = Bundle + code + ceremony date (Notes); storage = also share number
[ ] k-of-n (total + threshold) in the custody log ONLY, never on the cards
[ ] Wrapped seed imported to HSM, then file destroyed
[ ] Custody log filled; cards sealed in separate locations
[ ] Orchestrator reset; Admin card stored separately

Officer 1: _______________   Officer 2: _______________   Date: ___________
```

11.3 Annual verification

```
Year: _______
[ ] Collected cards (all n, or k + rotate): #____  #____  #____  #____
[ ] backup-test RESULT = PASS
[ ] Export counters checked against log
[ ] Cards returned & sealed; log updated

Signed: _______________   Date: ___________
```

11.4 Custodian change

```
Leaving: _____________   New holder: _____________   Card #____
[ ] Card retrieved
[ ] PIN reset with Admin card + new PIN set (or card re-provisioned)
[ ] Reassigned; custody log updated
[ ] backup-test PASSED

Signed: _________________   Date: ___________
```

11.5 Lost / damaged / locked card

```
Card #____   Issue: ______________________
[ ] Forgotten/locked PIN, card in hand -> reset with Admin card
[ ] Lost/destroyed -> usable cards remaining: ____  (need >= k = ____)
    [ ] >= k -> recovery OK; plan re-ceremony to restore redundancy
    [ ] <  k -> CANNOT reconstruct from cards; escalate immediately

Signed: _________________   Date: ___________
```

© 2026 Cryptnox SA

12 Security Model

This page is written for **risk, security, and compliance teams**. It describes what the system protects, what each actor can see, and the assumptions you must uphold for the guarantees to hold.

12.1 The core guarantee

The secret is split with Shamir's Secret Sharing into `n` shares with threshold `k`:

- **Any ``k`` shares** reconstruct the secret exactly.
- **Any ``k - 1`` shares** reveal **nothing** — this is information-theoretic, not merely "computationally hard". Fewer than `k` shares are as useless as zero shares.

On top of this, the hardware ensures the secret and the individual shares are **never present in plaintext on the host PC**.

12.2 What each actor can see

The host PC (running `shamir-cli`) only ever handles:

- non-secret public values exchanged between the cards,
- **encrypted** shares (unreadable outside the cards),
- the **RSA-wrapped** seed (readable only by the HSM's private key).

The host **never** sees the seed in plaintext, a share in plaintext, or any private key.

A single storage card holds exactly one **encrypted** share, behind a PIN.

The orchestrator card is the only place the seed exists in plaintext, and only transiently in volatile memory during a ceremony or recovery. It is erased on reset and after export.

12.3 What a stolen card reveals

Attacker has…	…and learns
One storage card — back markings	The bundle ID and a **share number** only. The total `n` and threshold `k` are **not** on the card (custody log only), so a found card does not reveal the scheme.
One storage card, no PIN	Nothing. The share is encrypted and PIN-gated.
One storage card, with PIN	One share — **useless alone**. `k` cards are still required.
`k - 1` storage cards, all PINs	**Nothing about the secret** (Shamir guarantee).
`k` storage cards, all PINs	The secret. This is the quorum you defined — protect at least `n - k + 1` cards from any single adversary.

> **Note**
>
> Choose `k` so that no single individual, site, or failure can assemble `k` cards. Distribute the cards across **independent** custodians and locations.

12.4 PIN protection

- Each storage card is protected by a **4-character PIN**.
- **Three** consecutive wrong PINs **lock** the card.
- Change a card's PIN with `shamir-cli storage change-pin`.
- The factory default PIN is `0000` — **change it** before going to production.

12.5 Card sets

The orchestrator and its storage cards form a **matched set**: an orchestrator only works with storage cards that belong to the same set, and rejects any card from outside it. This prevents an attacker from slipping a rogue storage card into a ceremony or recovery.

> **Note**
>
> In production, the cards you receive are already provisioned as a set — you do not configure this. Simply use the cards that were delivered together.

12.6 Seed wrapping (transport to the HSM)

When the orchestrator outputs the seed it always encrypts it with the HSM's **RSA public key**. Only the HSM holding the matching private key can open the resulting `wrapped_seed.bin`. Provide that key with `--rsa-key-file`.

> **Warning**
>
> If you omit `--rsa-key-file`, the tool generates a **throwaway** key for convenience during evaluation, and the wrapped file cannot be opened once the process exits. **Never** do this with a real secret — always supply your HSM's public key in production.

12.7 Audit: the export counter

Every real share read during a recovery increments a non-volatile **counter** inside the storage card. Inspect it:

```
shamir-cli storage read-counter
```

A counter higher than your records expect means a share was read outside a sanctioned recovery — investigate. (The *Backup Test* does **not** increment this counter, so routine checks do not pollute the audit trail.)

© 2026 Cryptnox SA

12.8 Assumptions you must uphold

For the guarantees above to hold:

1. The **ceremony host is trusted** and offline at ceremony time.
2. **Cards and PINs are distributed** so no single adversary can gather `k`.
3. The **HSM RSA public key** used for wrapping is authentic (verify its fingerprint out-of-band).
4. The **physical card labels** (share indices) are recorded and protected — recovery relies on inserting cards in share-index order (*Card order (share numbers)*).

See also

- *Concepts* — the HSM and wrapped seed in plain terms.
- *Choosing Parameters* — translating policy into `k` and `n`.
- *The Ceremony* — the operational controls in practice.

© 2026 Cryptnox SA

13 Command Reference

The `shamir-cli` command drives the whole system. This page lists every command. For guided procedures, see *The Ceremony*, *Recovery* and *Backup Test*.

13.1 Basic usage

```
shamir-cli [GLOBAL OPTIONS] GROUP COMMAND [ARGS]...
```

13.2 Global options

`-r, --reader` `TEXT`
: Select a reader by a case-insensitive substring of its name.

`-S, --simulator`
: Use the JCOP simulator on `localhost:8050` instead of a physical reader.

`--sim-port` `INTEGER`
: Simulator port (default `8050`).

`-v, --verbose`
: Print the raw APDU traffic, for debugging.

`-q, --quiet`
: Suppress non-essential output.

`--json`
: Emit machine-readable JSON, for scripting and integration.

13.3 Workflow commands

The high-level, guided procedures. **Start here.**

13.3.1 full-ceremony

Generate a seed (or import one), split it, and distribute shares to storage cards.

```
shamir-cli workflow full-ceremony -n N -k K -s SIZE [-t TYPE] \
    [--blessings-bundle FILE] [--rsa-key-file FILE] [--output-dir DIR]
    [--secret-file FILE | --secret HEX]
```

A real ceremony consumes the factory blessing bundle (`*.bundle.json`); `--blessings-bundle` is auto-discovered in the working dir if omitted. Only `--test-mode` runs without one.

Add `--secret-file` (or `--secret`) to **import** an existing secret instead of generating one on-card — same cards, same split, but the seed is not re-wrapped to an HSM. See *The Ceremony* for the full procedure.

13.3.2 full-recovery

Reconstruct the seed from `k` storage cards and re-wrap it for a new HSM.

```
shamir-cli workflow full-recovery -k K \
    [--rsa-key-file FILE] [--output-file FILE] [--pin PIN]
```

See *Recovery*.

13.3.3 backup-test

Verify that shares still reconstruct the seed, without exporting it.

```
shamir-cli workflow backup-test -k K [--pin PIN]
```

See *Backup Test*.

13.3.4 ceremony-import

Import an **existing** seed and back it up with Shamir (rather than generating a fresh one on-card). The secret is **required** — pass `--secret-file` (a hex or raw file) or `--secret` (hex string); its length must equal `-s`. There is no random fallback: for a fresh seed use `full-ceremony`.

```
shamir-cli workflow ceremony-import -n N -k K -s SIZE [-t TYPE] \
    (--secret-file FILE | --secret HEX) [--output-dir DIR]
```

13.4 Reader commands

13.4.1 list

List the connected smart card readers.

```
shamir-cli reader list
```

13.5 Orchestrator commands

These act on the orchestrator card. In normal use the **workflow** commands above drive it for you — the two commands below are the only ones you invoke directly.

13.5.1 info

Show the card type, firmware version, and state.

```
shamir-cli orchestrator info
```

13.5.2 reset

Reset the orchestrator to the initial state (`0x00`).

```
shamir-cli orchestrator reset
```

© 2026 Cryptnox SA

13.6 Storage commands

These act on a storage card.

13.6.1 info

Show the card type, version, and state.

```
shamir-cli storage info
```

13.6.2 reset

Reset the storage card to `0x00` (requires the PIN).

```
shamir-cli storage reset --pin 0000
```

13.6.3 verify-pin / change-pin

```
shamir-cli storage verify-pin [PIN]
shamir-cli storage change-pin [NEW_PIN]
```

13.6.4 read-counter

Read the export counter (for audit — see *Security Model*).

```
shamir-cli storage read-counter
```

13.6.5 pedersen-verify

Verify a single card — or a whole set with `--readers` — against its ceremony, without exporting the seed. See *Checking a single card*.

```
shamir-cli storage pedersen-verify
shamir-cli storage pedersen-verify --readers "<reader-1>,<reader-2>,..."
```

13.7 Pure-software Shamir (no card)

Run Shamir's maths entirely in software — useful for testing and for understanding the scheme. **No secret-bearing hardware is involved.**

13.7.1 split

```
shamir-cli shamir split <SECRET_HEX> -k K -n N -s SIZE
```

13.7.2 recover

```
shamir-cli shamir recover -s SIZE --share 1:HEX --share 2:HEX ...
```

© 2026 Cryptnox SA

13.7.3 verify

```
shamir-cli shamir verify <SECRET_HEX> -s SIZE --share 1:HEX ...
```

A concrete end-to-end example — split a 16-byte secret 2-of-3, then recover it from any two shares:

```
$ shamir-cli shamir split 000102030405060708090a0b0c0d0e0f -k 2 -n 3 -s 16
  Share  1: 555f6ab7fdfe2769ceb6da59773e816d
  Share  2: aabdd36cf7f748cc9564aaa7e26ff4cb
  Share  3: 001c3c21f1f06a2f5c127af64da16b46

$ shamir-cli shamir recover -s 16 \
    --share 1:555f6ab7fdfe2769ceb6da59773e816d \
    --share 3:001c3c21f1f06a2f5c127af64da16b46
  Recovered secret: 000102030405060708090a0b0c0d0e0f
```

Note

These commands never touch a card and are safe to experiment with. Do not use them on a **real** secret on a networked machine — the secret is handled in plaintext in the host's memory, unlike the hardware workflows.

See also

- *The Ceremony*, *Recovery*, *Backup Test* — the guided procedures built on these commands.

© 2026 Cryptnox SA

14 Troubleshooting

This page lists the situations you are most likely to hit and how to resolve them. Status words (`SW=...`) are the two-byte codes returned by the card.

14.1 No readers found

"No smart card readers found."

- Confirm the reader is connected: `shamir-cli reader list`.
- On Linux, ensure `pcscd` is running: `sudo systemctl status pcscd`.
- You may need permissions — run under `sudo` or add your user to the `pcscd` group.
- Remember: **only contact readers** work. NFC / contactless is rejected.

14.2 Command not allowed (`SW=6985`)

The card's state machine refused the command in its current state.

- You issued a step out of order (e.g. exporting a share before the seed was generated).
- The simplest fix is to **reset and start the workflow again**:

```
shamir-cli orchestrator reset
```

> **Note**
>
> The guided workflows (*The Ceremony*, *Recovery*) drive the state machine for you. If you see `6985` while using them, the card was likely left in a partial state — reset it and re-run.

14.3 Wrong data / crypto error (`SW=6A80`)

A cryptographic check failed. Common causes:

- The cards do **not** belong to the same set (a storage card from another set, or mixed with another orchestrator).
- The wrong **RSA key** was supplied for wrapping.

Re-run the workflow from a clean reset, using cards from the same set.

14.4 Wrong PIN

``*SW=63CX*`` — *"Wrong PIN, X tries remaining".*

- `X` is the number of attempts left before lockout.
- After **three** wrong attempts, the card locks.

- The factory default PIN is `0000`.

> **Warning**
>
> A locked storage card cannot be read with its PIN. With an **Admin card** (PIN level C) you can reset it; otherwise the three-attempt budget is precious — keep PINs in your custody records. See *Storage card PIN — security levels*.

14.5 Resetting a stuck card

If a card is left in a non-initial state (for example after an interrupted ceremony), simply reset it:

```
# Orchestrator
shamir-cli orchestrator reset

# Storage (PIN required)
shamir-cli storage reset --pin 0000
```

If a storage card's PIN is lost or locked (after three wrong attempts):

- **With an Admin card (PIN level C)** — reset the card's PIN with the Admin card and set a new one. The card and its share are recovered. This is the recommended setup (see *Storage card PIN — security levels*).
- **Without an Admin card (level A/B)** — the card cannot be unlocked; it, and the share it holds, are lost. As long as you still have `k` working cards, recovery is unaffected; if you drop below `k`, the secret can no longer be reconstructed from the cards.

> **Note**
>
> The **orchestrator** is different: it holds **no secret** at rest. If an orchestrator card is lost, damaged, or locked, it is not a security concern — request a replacement orchestrator card (matched to your card set) from Cryptnox at contact@cryptnox.com. Your storage cards and their shares are unaffected.

14.6 Cards still "Loaded" before a new ceremony

If a storage card reports state `0x02` (Loaded), it still holds a share from a previous ceremony. Reset it first:

```
shamir-cli storage reset --pin 0000
```

14.7 Recovery produced the wrong secret

The recovery completed but the `Recovery checksum` does **not** match the checksum recorded at the ceremony.

- **Cause:** the cards were inserted in the wrong order, or cards from a different ceremony were mixed in. The system relies on inserting cards in **share-index order** (*Card order*

(share numbers)).

- **Fix:** do not import the result. Re-run, inserting the correctly labelled cards (#1 first, #2 second, . . .).

Important

Always verify the `Recovery checksum` against your ceremony record **before** importing a recovered seed into an HSM.

14.8 Reconstruction is slow

On-card reconstruction performs modular inversions, which are expensive on a smart card. A large threshold on a large curve (for example `k = 7` on P-521) can take on the order of a second. This is expected.

See also

- *Recovery* — the correct recovery procedure.
- *Security Model* — PINs, lockout, and the audit counter.

© 2026 Cryptnox SA

15 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

The `shamir-cli` software and the card applets are proprietary to Cryptnox SA and are provided under a commercial license agreement; this documentation license does not cover them.

For inquiries, contact: contact@cryptnox.com

```
Attribution-NonCommercial-NoDerivatives 4.0 International

=======================================================================

Creative Commons Corporation ("Creative Commons") is not a law firm and
does not provide legal services or legal advice. Distribution of
Creative Commons public licenses does not create a lawyer-client or
other relationship. Creative Commons makes its licenses and related
information available on an "as-is" basis. Creative Commons gives no
warranties regarding its licenses, any material licensed under their
terms and conditions, or any related information. Creative Commons
disclaims all liability for damages resulting from their use to the
fullest extent possible.

Using Creative Commons Public Licenses

Creative Commons public licenses provide a standard set of terms and
conditions that creators and other rights holders may use to share
original works of authorship and other material subject to copyright
and certain other rights specified in the public license below. The
following considerations are for informational purposes only, are not
exhaustive, and do not form part of our licenses.

     Considerations for licensors: Our public licenses are
     intended for use by those authorized to give the public
     permission to use material in ways otherwise restricted by
     copyright and certain other rights. Our licenses are
     irrevocable. Licensors should read and understand the terms
     and conditions of the license they choose before applying it.
     Licensors should also secure all rights necessary before
     applying our licenses so that the public can reuse the
     material as expected. Licensors should clearly mark any
     material not subject to the license. This includes other CC-
     licensed material, or material used under an exception or
```

```
    limitation to copyright. More considerations for licensors:
   wiki.creativecommons.org/Considerations_for_licensors

    Considerations for the public: By using one of our public
    licenses, a licensor grants the public permission to use the
    licensed material under specified terms and conditions. If
    the licensor's permission is not necessary for any reason--for
    example, because of any applicable exception or limitation to
    copyright--then that use is not regulated by the license. Our
    licenses grant only permissions under copyright and certain
    other rights that a licensor has authority to grant. Use of
    the licensed material may still be restricted for other
    reasons, including because others have copyright or other
    rights in the material. A licensor may make special requests,
    such as asking that all changes be marked or described.
    Although not required by our licenses, you are encouraged to
    respect those requests where reasonable. More considerations
    for the public:
   wiki.creativecommons.org/Considerations_for_licensees

=======================================================================

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.

=======================================================================
```

```
Creative Commons is not a party to its public
licenses. Notwithstanding, Creative Commons may elect to apply one of
its public licenses to material it publishes and in those instances
will be considered the “Licensor.” The text of the Creative Commons
public licenses is dedicated to the public domain under the CC0 Public
Domain Dedication. Except for the limited purpose of indicating that
material is shared under a Creative Commons public license or as
otherwise permitted by the Creative Commons policies published at
creativecommons.org/policies, Creative Commons does not authorize the
use of the trademark "Creative Commons" or any other trademark or logo
of Creative Commons without its prior written consent including,
without limitation, in connection with any unauthorized modifications
to any of its public licenses or any other arrangements,
understandings, or agreements concerning use of licensed material. For
the avoidance of doubt, this paragraph does not form part of the
public licenses.

Creative Commons may be contacted at creativecommons.org.
```