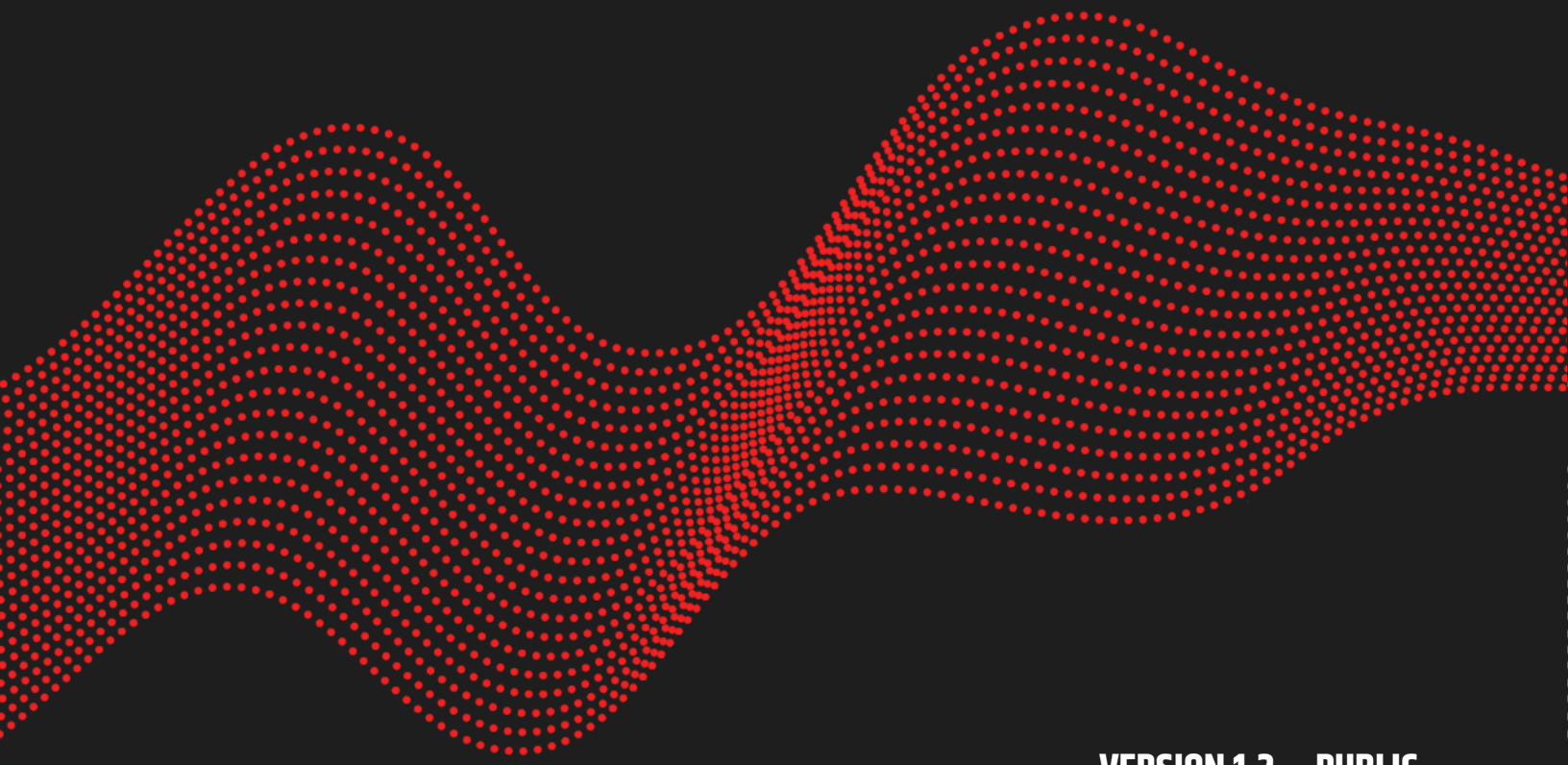
SYNACKTIV

LEDGER — SECURITY ASSESSMENT REPORT

CHARON - PENTEST

2025/05/02



VERSION 1.2 — PUBLIC

Contents

Introduction

Context and objectives

Ledger has asked Synacktiv to perform penetration tests on the Charon smart card as part of their pre-release security campaign. This smart card is based on the Javacard and Global Platform frameworks and developed by Ledger.

The tests were performed in three steps, from a black-box to white-box approach.

The primary objective of this audit was to evaluate the firmware's security posture, specifically focusing on:

- **`Extraction of Sensitive Data:`** Determining whether the firmware contains vulnerabilities which can allow exporting the master seed, or the user PIN code.
- **`Bypassing User Consent Mechanisms:`** Assessing if there are methods to bypass user consent to perform unauthorized actions on the device, including OS updates, application data export, or service activation.

Scope and limits

The penetration tests scope only includes the following assets:

Assets	
Applet	commit ca5b840
BOLOS code	v1.3.0-rc2

Black-box	Grey-box	White-box
The public applet sources and iso-prod devices are provided	Charon devices and ISD/SSD and SCP keys are provided	Anything needed in order to carry out in-depth tests

Timeline

The security assessment was performed from the Synacktiv offices from the 14th of April to the 5th of May 2025.

Date	Description
2025/04/14	Kick-off
2025/04/14	Beginning of the tests
2025/04/23	Follow-up meeting
2025/05/02	End of the tests

Version history

Version	Comment
v1.0	Initial version
v1.1	Update the Security Domain chapter
v1.2	Update details in the Security Domain chapter

Metrics

Security level rating

Synacktiv experts determine a global security level of the audited target given the audited scope, corresponding observations and state of the art.

HIGH

No sensitive asset was compromised or impacted during the audit period. A few improvements were nonetheless identified.

SATISFACTORY

No sensitive asset was compromised or impacted during the audit period. However, several identified vulnerabilities may help to impact these assets given specific prerequisites (publication of a new vulnerability, different network setup, etc.).

INSUFFICIENT

Some attack vectors impacting sensitive assets were identified during the audit period.

LOW

Sensitive assets of the audited scope were compromised during the audit period.

Vulnerability rating

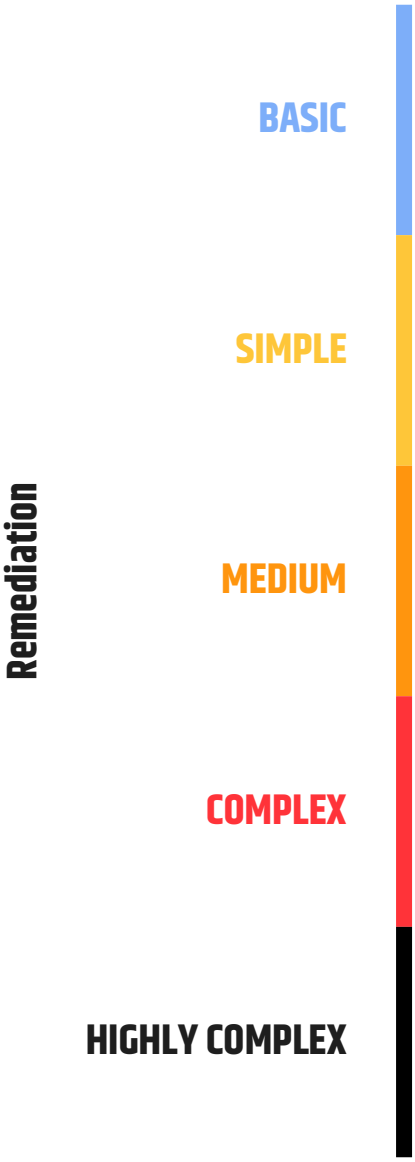
Synacktiv experts classify the sensitivity of the identified vulnerabilities and determine a grade of Severity (S), resulting from the product of two intermediate scores Probability (P), and Impact (I).

This scoring system is close to the concept of probabilistic risk assessment used in the industrial sector.

Probability	RARE	Hidden attack vector and/or needing high prerequisites hard to obtain.
	LOW	Vulnerability difficult to identify, the attacker must have technical information on the target or must exploit intermediate vulnerabilities.
	MEDIUM	Vulnerability identifiable by an average attacker.
	HIGH	Vulnerability easy to identify by an attacker, attack vector accessible without any particular constraint.
	FREQUENT	Vulnerability trivial to identify and potentially already identified.
Impact	MINIMAL	Exploitation of the vulnerability makes it possible to obtain non-sensitive technical information on the target.
	LOW	Exploiting the vulnerability provides technical information about the target.
	MEDIUM	The vulnerability allows an attacker to partially compromise the security of the target.
	HIGH	The attacker can access and/or modify sensitive information compromising the security of the target and its environment.
	MAXIMAL	The attacker can compromise the majority of the information system or the most sensitive data through the vulnerability.
Severity	REMARK	Negligible risk, non-compliance with hardening procedures. The vulnerability does not pose a significant risk to the target.
	LOW	Vulnerability remediation is used to comply with good security practices.
	MEDIUM	Vulnerability presents a risk to the target and needs to be fixed in the short term.
	HIGH	Vulnerability presents a significant risk for the target and must be fixed in the very short term.
	CRITICAL	Vulnerability presents a major risk for the target and requires immediate consideration.

Remediation rating level

Synacktiv provides an indicative level of complexity for vulnerability remediation. Due to limited visibility across the entire information system, this level may differ from the actual complexity of remediation.



BASIC

Instant, simple remediation requiring no specific security skills and without side effects.

SIMPLE

Low-complexity remediation that can be quickly performed by the development or administration team of the information system without side effects.

MEDIUM

Remediation requiring a preliminary study and a minimal monitoring of implemented measures by the administration team of the information system.

COMPLEX

High-complexity remediation involving structural changes to the information system and requiring an implementation by a dedicated team.

HIGHLY COMPLEX

A highly complex remediation involving a partial overhaul of the information system as well as a substantial human and financial investment.

Executive summary

Global security level

The security assessment performed by Synacktiv on Charon revealed a high security level.

Indeed, no scenarios have been identified that would allow extraction of the user's master seed or PIN without their consent.

In addition, while a few bugs were identified, which can lead to device resets. These issues do not fall within the scope of Ledger's defined threat model and do not pose a risk to the confidentiality of sensitive user data.

Synacktiv identified 0 security issues: .

Strengths

No critical vulnerabilities were found that would allow to bypass the seed-access protection mechanisms. The product leverages the security features of the Java Card SDK and GlobalPlatform, while adding an extra layer of asymmetric encryption for communication with the `SCP Ledger` protocol. It also mitigates several attack scenarios, including card removal.

Weaknesses

One area for improvement is the lack of a mechanism to verify the authenticity of the installed application against its open-source counterpart.

Including a cryptographic hash or digital signature of the applet in the `INS_GET_STATUS` response, would allow users to verify that the installed applet matches the open-source version, thereby enhancing transparency and trust.

Assessment details

This assessment report is the result of a three weeks study which involved two Synacktiv experts. Ledger provided an archive containing source code of the Java applet running on the Charon card and also one for the Bolos source code running on the wallet. Synacktiv team reviewed the source code and performed testing against Ledger devices (2 Flex, 4 Charon and 2 development cards).

An audit of the BOLOS code handling Charon card communication was performed over a two-day period. No vulnerabilities were identified during this review. However, the constrained time frame did not allow for exhaustive analysis of every component. Synacktiv consultants recommend allocating a dedicated assessment on this interface.

Command Processing

The `AppletCharon.process` method handles incoming Application Protocol Data Units (APDU) when the applet is selected, supporting three command classes:

- `GP_CLA_INITIALIZE_UPDATE` (`0x80`): Global Platform initialization.
- `GP_CLA_EXTERNAL_AUTHENTICATE` (`0x84`): Global Platform authentication.
- `LEDGER_COMMAND_CLA` (`0x08`): Ledger-specific commands.

For example, the following plaintext APDU retrieves card information after AppletCharon has been selected:

```
08f2000000
 ^ ^ ^ ^ ^
 | | | | |
 | | | | -> LC
 | | | -> P2
 | | -> P1
 | -> INS (INS_GET_STATUS)
 -> CLA (LEDGER)
```

Global Platform commands include:

- `GP_INS_INITIALIZE_UPDATE` (`0x50`): Initiates secure channel setup.
- `GP_INS_EXTERNAL_AUTHENTICATE` (`0x82`): Authenticates external entities.

The `CommandProcessor` class processes the Ledger commands while enforcing specific state requirements via the `AppletStateMachine` (persistent) and `TransientStateMachine` (transient).

The table below outlines commands, their prerequisites, and state transitions.

Command (INS)	AppletStateMachine State	TransientStateMachine State	PINManager State	SeedManager State	Success FSM Transitions	Error FSM Transitions
INS_GET_STATUS	Any	Any	Any	Any	None	None
INS_GET_PUBLIC_KEY	STATE_FABRICATION	STATE_IDLE	Any	Any	None	None
INS_SET_CERTIFICATE	STATE_FABRICATION	STATE_IDLE	Any	Any	EVENT_SET_CERTIFICATE (AppletFSM)	None
INS_SET_STATUS	STATE_PENDING_TESTS	STATE_IDLE	Any	Any	EVENT_FACTORY_TESTS_PASSED (AppletFSM), EVENT_SET_CERTIFICATE_AND_TESTS_PASSED (TransientFSM)	None
INS_GET_CARD_CERTIFICATE	STATE_ATTESTED or STATE_USER_PERSONALIZED	STATE_INITIALIZED or STATE_PIN_LOCKED	Any	Any	None	None
INS_VALIDATE_HOST_CERTIFICATE	STATE_ATTESTED or STATE_USER_PERSONALIZED	STATE_INITIALIZED or STATE_PIN_LOCKED	Any	Any	EVENT_CERT_VALID (TransientFSM, for ephemeral certificate)	None
INS_SET_PIN	STATE_ATTESTED	STATE_AUTHENTICATED	PIN_STATUS_NOT_SET	Any	PIN_STATUS_SET (PINManager)	None
INS_VERIFY_PIN	STATE_USER_PERSONALI	STATE_AUTHENTICATED	PIN_STATUS_ACTIVATED	Any	EVENT_PIN_VERIFIED	EVENT_PIN_TRY_LIMIT_EXCEEDED

Command (INS)	AppletStateMachine State	TransientStateMachine State	PINManager State	SeedManager State	Success FSM Transitions	Error FSM Transitions
	ZED				(TransientFSM)	(AppletFSM and TransientFSM)
INS_PIN_CHANGE	STATE_USER_PERSONALIZED	STATE_PIN_UNLOCKED	PIN_STATUS_ACTIVATED	Any	None	None
INS_SET_SEED	STATE_ATTESTED	STATE_AUTHENTICATED	PIN_STATUS_SET	seedSet == false	EVENT_SET_SEED (AppletFSM), EVENT_PIN_VERIFIED (TransientFSM), seedSet = true (SeedManager), PIN_STATUS_ACTIVATED (PINManager)	None
INS_RESTORE_SEED	STATE_USER_PERSONALIZED	STATE_PIN_UNLOCKED	Any (nothing explicit)	seedSet == true	None	None
INS_VERIFY_SEED	STATE_USER_PERSONALIZED	STATE_PIN_UNLOCKED	Any (nothing explicit)	seedSet == true	None	None
INS_GET_DATA	STATE_USER_PERSONALIZED	>= STATE_AUTHENTICATED	Any	Any	None	None
INS_SET_DATA	STATE_USER_PERSONALIZED	STATE_PIN_UNLOCKED	Any	Any	None	None
INS_FACTORY_RESET	STATE_USER_PERSONALI	STATE_PIN_UNLOCKED	PIN_STATUS_ACTIVATED	Any	EVENT_FACTORY_RESET (AppletFSM),	None

Command (INS)	AppletStateMachine State	TransientStateMachine State	PINManager State	SeedManager State	Success FSM Transitions	Error FSM Transitions
	ZED				EVENT_FACTORY_RESET (TransientFSM)	
INS_REQUEST_UPGRADE	STATE_USER_PERSONALIZED	STATE_AUTHENTICATED or STATE_PIN_UNLOCKED	PIN_STATUS_ACTIVATED	Any	upgradeAuthorizationState[0] = UPGRADE_AUTHORIZATION_GRANTED	upgradeAuthorizationState[0] = UPGRADE_AUTHORIZATION_DENIED

Sensitive operations like `INS_RESTORE_SEED` and `INS_VERIFY_SEED` mandate `STATE_PIN_UNLOCKED`, ensuring PIN verification precedes seed access.

Errors trigger an `ISOException` (e.g., `SW_CONDITIONS_NOT_SATISFIED` (`0x6985`) or `SW_WRONG_DATA` (`0x6A80`)), halting execution without unintended state changes.

Atomicity

Critical operations (e.g., `INS_SET_PIN`, `INS_SET_SEED`) uses `JCSystem` transactions (with `JCSystem.beginTransaction()`, `JCSystem.commitTransaction()` and `JCSystem.abortTransaction()`) to ensure atomic updates, mitigating risks from power loss or card detachment. If interrupted, the transaction is aborted, preventing partial updates and maintaining security. Post-transition checks, such as state validation in `AppletStateMachine.isValidState`, `TransientStateMachine.isValidState` and `PINManager.isValidState` methods, also ensure that only valid states are set.

However, `PINManager.changePIN` lacks explicit transaction wrapping, posing a potential issue if interrupted.

State Machines

The applet leverages three state machines that ensure sensitive operations such as seed access and PIN change are properly authenticated.

AppletStateMachine

The `AppletStateMachine` defines persistent states:

- `STATE_FABRICATION`: Initial fabrication phase.
- `STATE_PENDING_TESTS`: The certificate is loaded and tests are pending.
- `STATE_ATTESTED`: The certifcate and tests have been passed.
- `STATE_USER_PERSONALIZED`: The PIN and seed are fully configured.

Transitions are strictly controlled:

- `EVENT_SET_CERTIFICATE`: `STATE_FABRICATION` -> `STATE_PENDING_TESTS`
- `EVENT_FACTORY_TESTS_PASSED`: `STATE_PENDING_TESTS` -> `STATE_ATTESTED`
- `EVENT_SET_SEED`: `STATE_ATTESTED` -> `STATE_USER_PERSONALIZED`, but only if both PIN is activated and seed is set (checked in AppletCharon.onConsolidate).
- `EVENT_PIN_TRY_LIMIT_EXCEEDED`: `STATE_USER_PERSONALIZED` -> `STATE_ATTESTED`
- `EVENT_FACTORY_RESET`: `STATE_USER_PERSONALIZED` -> `STATE_ATTESTED`

TransientStateMachine

The `TransientStateMachine` manages transient states:

- `STATE_IDLE`: Fabrication default.
- `STATE_INITIALIZED`: The applet has been attested (certificate and tests passed).
- `STATE_PIN_LOCKED`: The seed and PIN code are set, but the PIN has not been verified.
- `STATE_AUTHENTICATED`: The host certificate has been validated.
- `STATE_PIN_UNLOCKED`: The PIN code has been verified.

Transitions require specific events:

- `EVENT_SET_CERTIFICATE_AND_TESTS_PASSED`: `STATE_IDLE` -> `STATE_INITIALIZED`
- `EVENT_CERT_VALID`: `STATE_INITIALIZED` -> `STATE_AUTHENTICATED`
- `EVENT_CERT_VALID`: `STATE_PIN_LOCKED` -> `STATE_AUTHENTICATED`
- `EVENT_PIN_VERIFIED`: `STATE_AUTHENTICATED` -> `STATE_PIN_UNLOCKED`
- `EVENT_PIN_TRY_LIMIT_EXCEEDED`: `STATE_AUTHENTICATED` -> `STATE_INITIALIZED`
- `EVENT_APPLET_DESELECTED` (with `AppletStateMachine.STATE_USER_PERSONALIZED`): `STATE_PIN_UNLOCKED` -> `STATE_PIN_LOCKED`
- `EVENT_APPLET_DESELECTED` (without `AppletStateMachine.STATE_USER_PERSONALIZED`): `STATE_PIN_UNLOCKED` -> `STATE_INITIALIZED`
- `EVENT_FACTORY_RESET`: `STATE_PIN_UNLOCKED` -> `STATE_INITIALIZED`

The `TransientStateMachine.setOnSelectState` method defines the state of the transient state machine depending on the applet state machine:

- `AppletStateMachine.STATE_FABRICATION`: `STATE_IDLE`
- `AppletStateMachine.STATE_PENDING_TESTS`: `STATE_IDLE`
- `AppletStateMachine.STATE_ATTESTED`: `STATE_INITIALIZED`
- Default: `STATE_PIN_LOCKED`

PINManager state

The `PINManager` state machine manages the PIN code status:

- `PIN_STATUS_NOT_SET`: default state with no PIN code configured.
- `PIN_STATUS_SET`: The PIN code has been created, but is not usable (the device lacks seed).

- `PIN_STATUS_ACTIVATED`: The PIN code and seed have been set -> the PIN code can be used to access the seed.
- `PIN_STATUS_INVALID`: This code is not used in the code.

Transitions are strictly controlled:

- `PINManager.createPIN()`: `PIN_STATUS_NOT_SET` -> `PIN_STATUS_SET`
- `PINManager.activatePIN()`: `PIN_STATUS_SET` -> `PIN_STATUS_ACTIVATED`
- `PINManager.unsetPIN()`: `PIN_STATUS_SET` -> `PIN_STATUS_NOT_SET`
- `PINManager.resetPIN()`: `PIN_STATUS_ACTIVATED` -> `PIN_STATUS_NOT_SET`

PIN and Seed Management

The applet uses secure channels (`SCP03` and `Ledger SCP`) for communication, manages states to control access, and clears data on errors, adding layers to protect both seed and PIN. State machines require proper steps, like certificate validation and PIN entry, to access sensitive data, without any shortcuts found.

PIN

The applet manages its PIN through the `PINManager` class, which relies on the `javacard.framework.OwnerPIN` class from the Java Card framework to securely store and verify the PIN.

The `PINManager.verifyPIN` method checks the PIN by first confirming that the PIN status is set to `PIN_STATUS_ACTIVATED`. If this condition is not met, the method throws an `ISOException` with the status code `SW_CONDITIONS_NOT_SATISFIED`.

Once the status is verified, the method retrieves the PIN length from the input buffer and uses `OwnerPIN.check()` to compare the provided PIN against the stored value, returning a boolean result to indicate success or failure.

The PIN length is restricted to between `4` and `8` bytes, as defined by the constants `PIN_MIN_SIZE` and `PIN_MAX_SIZE`, and users are limited to a maximum of `3` verification attempts to protect against brute-force attacks.

If this limit is exceeded, the applet clears both the seed and the PIN to prevent unauthorized access.

The `PINManager.changePIN` method allows users to update the PIN, but this is only permitted when the applet is in the `STATE_PIN_UNLOCKED` state, meaning the current PIN has already been successfully verified.

This restriction ensures that only authenticated users can modify the PIN.

Seed

The seed, a crucial element for cryptographic key derivation, is managed by the `SeedManager` class and stored as an AES-256 key within an `AESKey` object.

To protect the integrity of the seed, the applet computes an `AES-CMAC-128` using a predefined message (`SEED_CHECK_MSG`), enabling detection of any tampering or corruption.

The seed can only be set using the `INS_SET_SEED` command, which requires the applet to be in both the `STATE_ATTESTED` and `STATE_AUTHENTICATED` states, with the PIN already established.

Once set, the seed can only be accessed through specific commands like `INS_RESTORE_SEED` and `INS_VERIFY_SEED`, both of which require the applet to be in the `STATE_PIN_UNLOCKED` state, enforcing PIN verification before any seed-related operations.

In critical situations, such as exceeding the PIN attempt limit or performing a factory reset, the `clearSeed` method is triggered. This method overwrites the seed with random data and resets the `seedSet` flag to `false`, ensuring the original seed cannot be recovered.

This process follows best practices for securely erasing sensitive data in cryptographic systems.

Buffer Management

The applet uses transient buffers, such as `ramBuffer`, to manage temporary data during APDU processing and secure channel operations. These buffers are created as transient arrays using `JCSystem.makeTransientByteArray()` with the `CLEAR_ON_DESELECT` option, which automatically clears their contents when the applet is deselected.

This feature prevents sensitive data like decrypted seeds or PINs from persisting across sessions, reducing the risk of unauthorized access to residual information.

The applet takes additional steps to erase sensitive data from buffers after use. For instance, after handling private keys or other critical data, the applet fills buffers with `0xFF`. This combination of automatic clearing upon deselection and explicit erasure provides strong protection against data leakage.

Secure Channel

The applet employs secure channels, specifically `SCP03` and `Ledger SCP`, to protect data during transmission. These channels encrypt and decrypt APDU data, ensuring that sensitive information, such as the seed or PIN, is never sent in plain text.

The `secureChannel.unwrap` method decrypts incoming data, while the `secureChannel.wrap` method encrypts outgoing responses. This setup ensures that even if an attacker intercepts the communication, they cannot access the data without breaking the secure channel cryptographic protections.

Applet Upgrade

The upgrade process for Applet Charon involves updating the applet's code to a new version while preserving critical data, such as the seed and PIN. The new applet code is delivered in an ELF (Executable and Linkable Format) file..

The `INS_REQUEST_UPGRADE` command handles this by setting the `upgradeAuthorizationState` to `UPGRADE_AUTHORIZATION_GRANTED` once the PIN is verified.

Without this authorization, the upgrade cannot proceed, ensuring that only authenticated users can initiate the process.

When Applet Charon is upgraded, the new version inherits the same **`Application Identifier`** (AID) and context as the original. This allows the upgraded applet to access the same data objects (e.g., PIN, seed) without breaching the firewall, ensuring continuity while maintaining isolation from other applets. While this is a necessary part of the upgrade process, it introduces a risk if the upgrade isn't properly secured. However, the requirement for PIN verification before authorizing an upgrade mitigates this risk, as only an authorized user can start the process.

To improve security and transparency, one could benefit from an additional information field in the `INS_GET_STATUS` response, such as the applet signature. This would help users to confirm that the installed applet matches the version they've reviewed (e.g., on GitHub), offering additional assurance against unauthorized or malicious upgrades.

Security Domain

The **Secondary Security Domain** (SSD) named **SSD_LEDGER**, identified by the **Application Identifier** (AID) **5353445F4C4544474552**, is responsible for managing the Charon applet with specific privileges granted by the **Issuer Security Domain** (ISD). These privileges include:

- **SecurityDomain**: Manages its own domain and associated applets.
- **TrustedPath**: Establishes secure channels for communication within its domain.

Due to the absence of the **Authorized Management** privilege, the SSD cannot independently perform an **ELF_UPGRADE** as defined by GlobalPlatform Amendment H. Instead, Ledger carries out and **ELF_UPGRADE** via the SEMS service described in GlobalPlatform Amendment I. In this setup, the **upgradeAuthorizationState** variable, a transient short array used in the necessary hooks for an **ELF_UPGRADE**, determines whether the upgrade is authorized.

The upgrade process starts with an **INS_REQUEST_UPGRADE** command, including a PIN code. The applet must be in **STATE_USER_PERSONALIZED** and either **STATE_AUTHENTICATED** or **STATE_PIN_UNLOCKED**, with the PIN in **PIN_STATUS_ACTIVATED**. Upon successful PIN verification, the **upgradeAuthorizationState** is set to **UPGRADE_AUTHORIZATION_GRANTED**, allowing the upgrade to proceed, while a failed verification leaves it as **UPGRADE_AUTHORIZATION_DENIED**, ensuring that only an authenticated user can initiate the upgrade.

Thus, the SSD requires the user to enter the correct PIN to authorize the upgrade, preserving the applet's existing data, such as the PIN and seed, so the upgraded applet can continue to access them seamlessly. While the ISD has the **Authorized Management** privilege and could potentially be used for initiating an **ELF_UPGRADE**, its secure-channel keys, are randomly generated, unique per device, and never stored. Furthermore, the ISD ELF_UPGRADE requires passing the authorization process, requiring a valid PIN code, making it unusable for this purpose.

SYNACKTIV

+33 1 45 79 74 75

contact@synacktiv.com

5 boulevard Montmartre

75002 – PARIS

www.synacktiv.com