SPECIFICATION

# Calypso Terminal API

## Calypso Crypto Symmetric

*Version 0.2.0-SNAPSHOT, 2026-07-20*

***Warning. The present document cancels and replaces the previous release.***

# Link and Contact

Website
https://calypsonet.org

Support
support@calypsonet.org

# Table of Contents

# 1. Abstract

This document is the official technical specification of the **Terminal Calypso Crypto Symmetric API** standardised by the **Calypso Networks Association** (CNA). It defines, in a language-agnostic way (using a notation inspired by the Kotlin language), the interfaces, classes, enumerations, errors, structural relationships and behavioural requirements that any conforming implementation of the Terminal Calypso Crypto Symmetric API MUST satisfy.

The Terminal Calypso Crypto Symmetric API is part of the broader CNA **Terminal API** family. It is the SPI through which the **Terminal Calypso Card API** delegates every symmetric-key cryptographic operation required to operate Calypso secure sessions, SV commands, PIN ciphering and key loading.

# Document Status

| | |
|---|---|
| **Reference** | YYMMDD-SP-CNATerminalAPI-CalypsoCryptoSymmetric |
| **Short name** | CNA-TCCS-API |
| **Version** | 0.2.0-SNAPSHOT |
| **Revision date** | 2026-07-20 |
| **Editor** | Calypso Networks Association |
| **Source repository** | https://github.com/calypsonet/calypsonet-terminal-calypso-crypto-symmetric-uml-api |
| **Reference license** | Creative Commons Attribution-NoDerivatives 4.0 International (CC BY-ND 4.0) |

The present document specifies the 0.2.0-SNAPSHOT of the Terminal Calypso Crypto Symmetric API. It defines a single, coherent baseline against which conforming implementations are evaluated; the **Since** column indicates the version in which each member was introduced.

# Revision List

This section lists the high-level changes per version. The complete, fine-grained changelog is maintained in the `CHANGELOG.md` file at the root of the source repository.

| Version / Date | Modifications |
|---|---|
| **v0.2.0-SNAPSHOT**<br>2026-07-20 | Baseline. |

# 2. Introduction

## 2.1. Purpose

The Terminal Calypso Crypto Symmetric API defines the contract that a **symmetric-key crypto module** MUST expose so that a Calypso transaction manager can delegate the cryptographic computations required by a Calypso card. The module is typically backed by a SAM (Secure Application Module) but the API is intentionally agnostic to the underlying implementation: any component capable of producing the required MACs, signatures and ciphered blocks MUST be acceptable.

The contract covers:

- the lifecycle of a Calypso secure session (initialisation, digest update, finalisation, mutual authentication),
- the encryption/decryption of session data,
- the computation of the security data exchanged during Stored Value (SV) operations,
- the ciphering of PIN values for presentation or modification,
- the generation of ciphered key blocks for card key loading.

## 2.2. Scope

This specification covers:

- the factory used by the upstream layer to instantiate a transaction-manager SPI,
- the transaction-manager SPI exposing the cryptographic primitives,
- the data carrier holding the computed SV security data,
- the errors raised by the SPI.

The following topics are **out of scope**:

- the wire-level dialog between the terminal and any underlying SAM,
- the management of key material,
- the cryptographic algorithms themselves—only the input/output contract is normative.

## 2.3. Conformance

A software product conforms to this specification if and only if:

1. it implements every interface and class defined in Chapter 5 with the operations and semantics prescribed in this document,
2. it raises errors exclusively of the types defined in Section 5.4 for the situations described,
3. it preserves the structural relationships described in Chapter 4,
4. every object it provides implements the marker interface of CNA-TCC-API that corresponds to its type, as described below.

The Calypso Card API (CNA-TCC-API) declares marker interfaces through which the objects of a symmetric crypto module are handed over to the card API. Those markers declare no operation: their content is the one defined by this specification. Every object implementing a type of the right column MUST therefore also implement the marker of the left column.

| CNA-TCC-API marker | CNA-TCCS-API type |
|---|---|
| `SymmetricCryptoCardTransactionManagerFactory` | `SymmetricCryptoCardTransactionManagerFactorySpi` |

Throughout this specification, the key words **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY** and **OPTIONAL** are to be interpreted as described in RFC 2119 and RFC 8174 when, and only when, they appear in all capitals.

In conformance with RFC 2119, **SHALL** is equivalent to **MUST**; this specification uses **MUST** exclusively in order to avoid ambiguity.

## 2.4. Document Conventions

### *2.4.1. Naming*

Type names follow `UpperCamelCase`; operation, parameter and enumeration-member names follow `lowerCamelCase`. SPI types are suffixed with `Spi`. Names defined by this specification are reproduced as-is in the UML class diagram (Section 4.3), which provides a visual representation of the specification.

### *2.4.2. Data types*

The following type names, whose notation is inspired by the Kotlin language, are used in operation signatures and are mapped, in each binding, to the closest equivalent native type. Implementations MUST preserve the semantic intent rather than the syntactic form.

| Type | Description | Typical bindings |
|---|---|---|
| `String` | Sequence of characters, UTF-8 encoded by default. | `String`, `string`, `str`. |
| `Boolean` | Two-state truth value. | `boolean`, `bool`. |
| `Byte` | Signed 8-bit value. | `byte`, `i8`. |
| `Int` | Signed 32-bit integer. | `int`, `Int32`, `i32`. |
| `ByteArray` | Ordered sequence of octets. | `byte[]`, `[u8]`, `bytes`. |
| `MutableList<T>` | Ordered collection of `T` values that the receiver may modify. | `ArrayList<T>`, `MutableList<T>`, `Vec<T>`. |
| `Any` | Value of any type, root of the type hierarchy. | `Object`, `Any`, `object`, `Box<dyn Any>`. |
| `T?` | Nullable value: either a `T` value or `null`. | `Optional<T>`, `T?`, `Option<T>`. |
| `null` | Absence of value, the alternative held by a nullable type. | `null`, `nil`, `None`. |
| `Unit` | Indicates that an operation returns no value. | `void`, `Unit`, `()`. |

### *2.4.3. Type tables*

Each interface, class, enumeration and error defined in this document is described by a table of the form:

| | |
|---|---|
| **Name** | The type's normalized name. |
| **Kind** | The category of the type (interface, class, enumeration, error, etc.). |
| **Namespace** | The namespace in which the type is defined. |
| **Extends** | The type extended by this type, when applicable. |
| **Implements** | The interface implemented by this type, when applicable. |
| **Since** | The version of the API in which the type was introduced. |

| | |
|---|---|
| **Purpose** | A normative description of the type's responsibility. |
| **See also** | Cross-references to related operations or types, each a hyperlink to the corresponding table. Present only when applicable. |

The **Purpose** row states, in one or two sentences, the responsibility of the type and the way an instance is obtained. It is meant to be scanned, not read: any content that requires structure or depth—rules, lists, notes, value tables or examples—is placed as prose below the table.

### *2.4.4. Operation tables*

Each operation defined in this document is described by a table of the form:

| | |
|---|---|
| **Signature** | The operation's signature. |
| **Since** | The version of the API in which the operation was introduced. |
| **Description** | A normative description of the operation's behaviour. |
| **Parameters** | A description of each input parameter. |
| **Returns** | A description of the returned value, if any. |
| **Pre-conditions** | The conditions the caller MUST satisfy before invoking the operation, each prefixed by its nature. |
| **Errors** | A list of error conditions a correct caller must handle, each mapped to a specific error type. |
| **See also** | Cross-references to related operations or types, each a hyperlink to the corresponding table. Present only when applicable. |

### *2.4.5. Operation contracts*

To avoid repetition, the following rules apply to every operation table in Chapter 5 and are therefore not restated for each operation:

- **Pre-conditions.** The **Pre-conditions** row states what the caller MUST guarantee before invoking the operation. Violating one is a contract error, never listed in the **Errors** row, which describes only the outcomes a correct caller must handle. Each entry is prefixed by its nature, which determines the error raised when the condition is not met:
  - **Argument**—an illegal argument error. Applies to any invalid argument, including values bounded by a card or SAM protocol.
  - **Range**—an index out of bounds error. Applies when the argument designates a position within a collection or memory image exposed by the API.
  - **State**—an illegal state error.
  - **Capability**—an unsupported operation error.
- **Non-`null` results.** Unless the return type is marked nullable (`T?`) or the **Returns** row states otherwise, an operation returns a non-`null` value.
- **Collections.** Unless the return type is marked nullable (`T?`), an operation whose return type is a collection (e.g. `List`, `Set`, `Map`) returns an empty collection rather than `null`.
- **Fluent composition.** Builder-style operations return the current instance so that calls can be chained.

### *2.4.6. Concurrency*

Unless explicitly stated otherwise, the stateful types defined by this specification are **not** thread-safe. A given instance MUST be accessed from a single thread at a time; concurrent use of the same instance without external synchronisation results in undefined behaviour. Distinct instances MAY be used concurrently.

# 2.5. References and Resources

## *2.5.1. Calypso References*

References to CNA Terminal APIs designate the indicated **major** version; any later release within the same major is backward-compatible per that API's evolution policy.

| Reference | Document |
|---|---|
| [CNA-TCC-API] | *CNA Terminal API — Calypso Card* (SP-CNATerminalAPI-CalypsoCard), version 3.0, Calypso Networks Association. |

## *2.5.2. Normative References*

| Reference | Document |
|---|---|
| [RFC 2119] | *Key words for use in RFCs to Indicate Requirement Levels*, S. Bradner, March 1997. |
| [RFC 8174] | *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*, B. Leiba, May 2017. |

## *2.5.3. External Resources*

| Resource | Link |
|---|---|
| Calypso Networks Association | https://calypsonet.org |
| Terminal APIs website | https://terminal-api.calypsonet.org |
| Terminal APIs documentation | https://docs.terminal-api.calypsonet.org |

# 3. Glossary and Acronyms

## 3.1. Glossary

| Term | Definition |
|---|---|
| Crypto Module | Component capable of performing the cryptographic operations described by this specification. Typically backed by a SAM but not required to be. |
| Extended Mode | Operating mode that supports Calypso Prime Extended products capabilities (such as APDU encryption/decryption, pre-open secure session, longer cryptographic data, etc.). |
| Secure Session | Calypso authenticated and integrity-protected exchange between the terminal and the card. |
| Session MAC | Message Authentication Code computed over the data exchanged during a Calypso secure session. |
| SV Command | Stored Value command (Load, Debit, Undebit) operating on a Calypso card. |
| SV MAC | Message Authentication Code protecting an SV command exchange. |
| SPI | Service Provider Interface — an interface designed to be implemented by an extension module. |

## 3.2. Acronyms

| Abbreviation | Expansion |
|---|---|
| APDU | Application Protocol Data Unit |
| CNA | Calypso Networks Association |
| KIF | Key Identifier |
| KVC | Key Version Code |
| MAC | Message Authentication Code |
| PIN | Personal Identification Number |
| SAM | Secure Application Module |
| SPI | Service Provider Interface |
| SV | Stored Value |
| UML | Unified Modelling Language |

# 4. Architectural Overview

## 4.1. Functional positioning

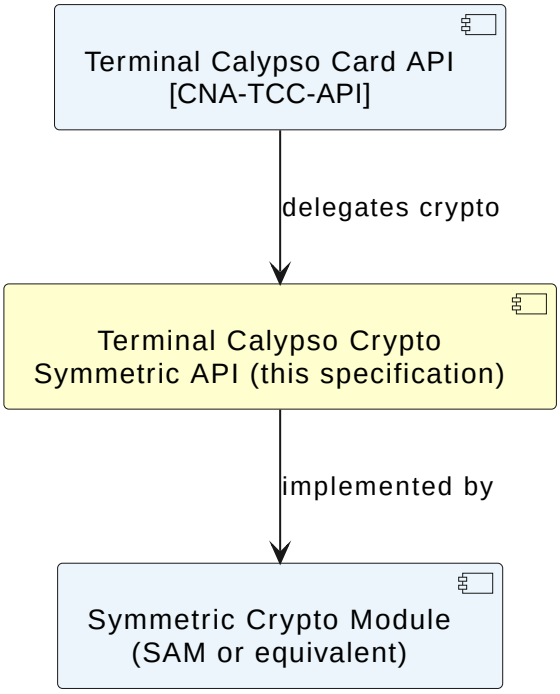


*Figure 1. Terminal Calypso Crypto Symmetric API — Architecture Overview*

## 4.2. Logical namespaces

| Namespace | Role | Summary |
|---|---|---|
| `calypso.crypto.symmetric` | Public API | Properties and errors consumed by both sides of the SPI. |
| `calypso.crypto.symmetric.spi` | SPI | Interfaces implemented by the symmetric crypto module, and the data they return. |

## 4.3. UML class diagram

The following UML class diagram provides a visual representation of the types and relationships defined by this specification:

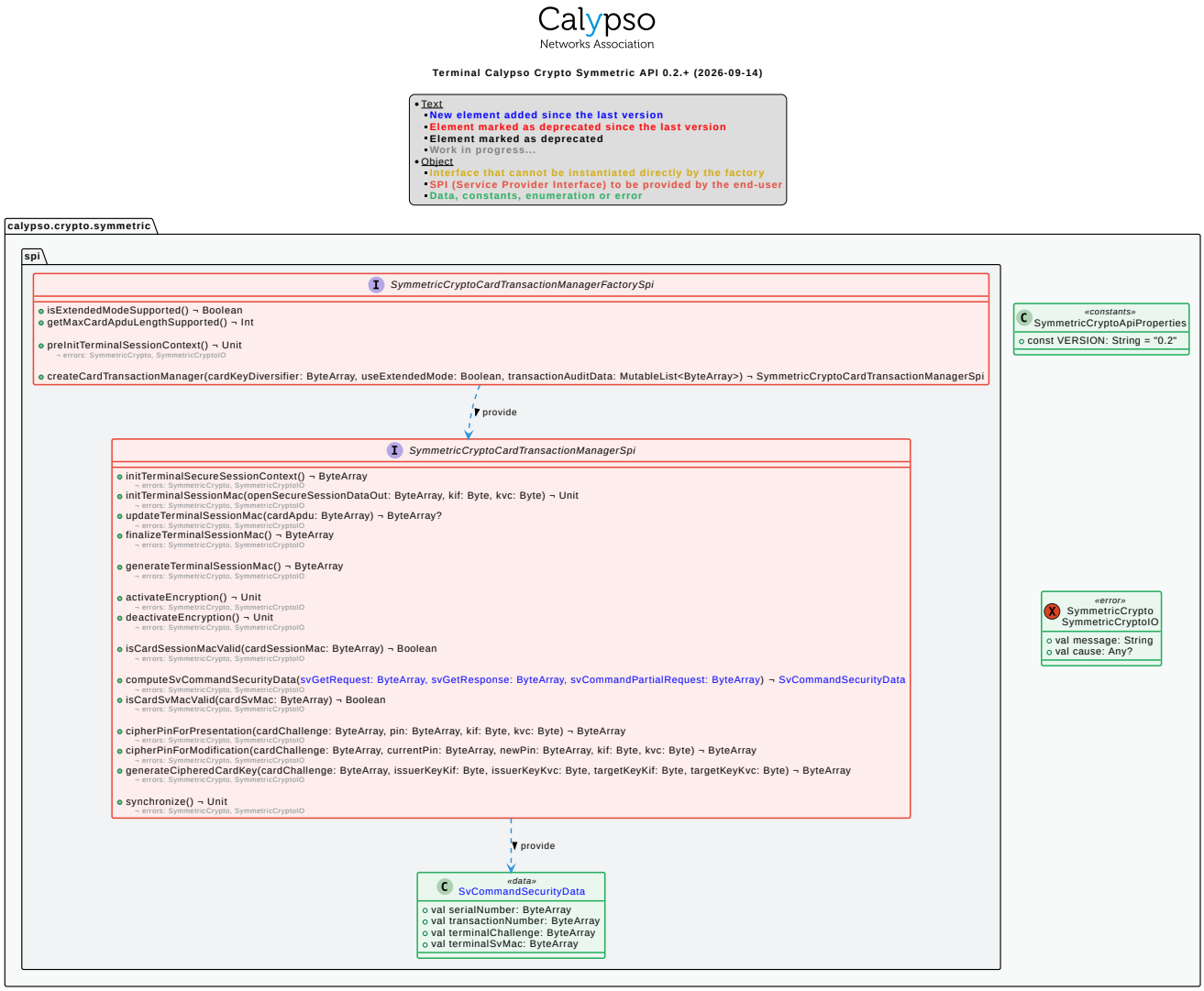


*Figure 2. Terminal Calypso Crypto Symmetric API — UML Class Diagram*

# 5. API Specification

## 5.1. Constants

### *5.1.1. Symmetric Crypto API Properties*

| | |
|---|---|
| **Name** | `SymmetricCryptoApiProperties` |
| **Kind** | Constants |
| **Namespace** | `calypso.crypto.symmetric` |
| **Since** | 0.1 |
| **Purpose** | Exposes the immutable properties of the Terminal Calypso Crypto Symmetric API. |

| Name | Type | Since | Description |
|---|---|---|---|
| `VERSION` | `String` | 0.1 | String representation of the version of the API implemented by the conforming binding. The value is a dotted decimal of the form `MAJOR.MINOR`; it is `"0.2"` for this revision of the specification. |

## 5.2. Data

### *5.2.1. SV Command Security Data*

| | |
|---|---|
| **Name** | `SvCommandSecurityData` |
| **Kind** | Data |
| **Namespace** | `calypso.crypto.symmetric.spi` |
| **Since** | 0.1 |
| **Purpose** | Carries the security data computed by the crypto module for an SV command (Load / Debit / Undebit), returned by `SymmetricCryptoCardTransactionManagerSpi.computeSvCommandSecurityData`. |

| Name | Type | Default | Since | Description |
|---|---|---|---|---|
| `serialNumber` | `ByteArray` | — | 0.1 | Serial number to be placed in the **SV Load/Debit/Undebit** command request. |
| `transactionNumber` | `ByteArray` | — | 0.1 | Transaction number to be placed in the **SV Load/Debit/Undebit** command request. |
| `terminalChallenge` | `ByteArray` | — | 0.1 | Terminal challenge to be placed in the **SV Load/Debit/Undebit** command request. |
| `terminalSvMac` | `ByteArray` | — | 0.1 | Terminal SV MAC to be placed in the **SV Load/Debit/Undebit** command request. |

## 5.3. SPI Interfaces

### *5.3.1. Symmetric Crypto Card Transaction Manager Factory SPI*

| | |
|---|---|
| **Name** | `SymmetricCryptoCardTransactionManagerFactorySpi` |
| **Kind** | Interface (SPI) |
| **Namespace** | `calypso.crypto.symmetric.spi` |

| | |
|---|---|
| **Since** | 0.1 |
| **Purpose** | Factory of `SymmetricCryptoCardTransactionManagerSpi`. Implemented by the crypto module. |

### *Create Card Transaction Manager*

| | |
|---|---|
| **Signature** | `createCardTransactionManager(`<br>`  cardKeyDiversifier: ByteArray,`<br>`  useExtendedMode: Boolean,`<br>`  transactionAuditData: MutableList<ByteArray>`<br>`) → SymmetricCryptoCardTransactionManagerSpi` |
| **Since** | 0.1 |
| **Description** | Creates a `SymmetricCryptoCardTransactionManagerSpi` providing the cryptographic primitives that a Calypso card transaction requires when using symmetric keys, initialised with the supplied card key diversifier and mode. Implemented by the crypto module. |
| **Parameters** | `cardKeyDiversifier` — the card key diversifier to use for the coming cryptographic computations.<br>`useExtendedMode` — a flag requesting the extended mode, if supported by the crypto service.<br>`transactionAuditData` — the list to which the crypto module appends the transaction audit data it records. |
| **Returns** | A `SymmetricCryptoCardTransactionManagerSpi`. |
| **Pre-conditions** | **Argument** — every parameter is non-`null`.<br>**State** — the extended mode is supported whenever it is requested. |
| **Errors** | None. |

### *Get Max Card APDU Length Supported*

| | |
|---|---|
| **Signature** | `getMaxCardApduLengthSupported() → Int` |
| **Since** | 0.1 |
| **Description** | Returns the maximum card APDU length supported by the crypto module. |
| **Returns** | A positive `Int`. |
| **Pre-conditions** | None. |
| **Errors** | None. |

### *Is Extended Mode Supported*

| | |
|---|---|
| **Signature** | `isExtendedModeSupported() → Boolean` |
| **Since** | 0.1 |
| **Description** | Indicates whether the **extended** mode is supported by the crypto module. |
| **Returns** | `true` if the extended mode is supported, `false` otherwise. |
| **Pre-conditions** | None. |
| **Errors** | None. |

### *Pre Init Terminal Session Context*

| | |
|---|---|
| **Signature** | `preInitTerminalSessionContext() → Unit` |
| **Since** | 0.1 |
| **Description** | Retrieves and stores the terminal challenge in the crypto module image for later use. |
| **Pre-conditions** | None. |

| | |
|---|---|
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### 5.3.2. Symmetric Crypto Card Transaction Manager SPI

| | |
|---|---|
| **Name** | `SymmetricCryptoCardTransactionManagerSpi` |
| **Kind** | Interface (SPI) |
| **Namespace** | `calypso.crypto.symmetric.spi` |
| **Since** | 0.1 |
| **Purpose** | Defines the cryptographic primitives required by a Calypso card transaction when using symmetric keys. An instance is obtained via `SymmetricCryptoCardTransactionManagerFactorySpi.createCardTransactionManager`. |

Every operation MUST raise either `SymmetricCrypto` (internal error) or `SymmetricCryptoIO` (communication-level error) when the cryptographic computation cannot be completed.

`finalizeTerminalSessionMac` and `isCardSessionMacValid` MUST be called only after every `updateTerminalSessionMac` call corresponding to the APDUs exchanged during the secure session.

#### *Activate Encryption*

| | |
|---|---|
| **Signature** | `activateEncryption() → Unit` |
| **Since** | 0.1 |
| **Description** | Activates the encryption/decryption of the data sent/received during the secure session. |
| **Pre-conditions** | None. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

#### *Cipher PIN For Modification*

| | |
|---|---|
| **Signature** | `cipherPinForModification(`<br>`  cardChallenge: ByteArray,`<br>`  currentPin: ByteArray,`<br>`  newPin: ByteArray,`<br>`  kif: Byte,`<br>`  kvc: Byte`<br>`) → ByteArray` |
| **Since** | 0.1 |
| **Description** | Computes a block of encrypted data to be sent to the card for a PIN modification.<br><br>**Note:** the `kif` and `kvc` parameters MUST be ignored when PIN modification is performed within a Secure Session. |
| **Parameters** | `cardChallenge` — a `ByteArray` containing the card challenge.<br>`currentPin` — a `ByteArray` containing the 4-byte current PIN value.<br>`newPin` — a `ByteArray` containing the 4-byte new PIN value.<br>`kif` — the PIN encryption key KIF.<br>`kvc` — the PIN encryption key KVC. |
| **Returns** | A non-empty `ByteArray` containing the encrypted data block to send to the card. |
| **Pre-conditions** | **Argument** — every parameter is non-`null`. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Cipher PIN For Presentation*

| | |
|---|---|
| **Signature** | `cipherPinForPresentation(`<br>`  cardChallenge: ByteArray,`<br>`  pin: ByteArray,`<br>`  kif: Byte,`<br>`  kvc: Byte`<br>`) → ByteArray` |
| **Since** | 0.1 |
| **Description** | Computes a block of encrypted data to be sent to the card for an enciphered PIN presentation.<br><br>**Note:** the `kif` and `kvc` parameters MUST be ignored when PIN verification is performed within a Secure Session. |
| **Parameters** | `cardChallenge` — a `ByteArray` containing the card challenge.<br>`pin` — a `ByteArray` containing the 4-byte PIN value.<br>`kif` — the PIN encryption key KIF.<br>`kvc` — the PIN encryption key KVC. |
| **Returns** | A non-empty `ByteArray` containing the encrypted data block to send to the card. |
| **Pre-conditions** | **Argument** — every parameter is non-`null`. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Compute SV Command Security Data*

| | |
|---|---|
| **Signature** | `computeSvCommandSecurityData(`<br>`  svGetRequest: ByteArray,`<br>`  svGetResponse: ByteArray,`<br>`  svCommandPartialRequest: ByteArray`<br>`) → SvCommandSecurityData` |
| **Since** | 0.1 |
| **Description** | Computes the security data needed to complete an **SV Load/Debit/Undebit** command from the supplied **SV Get** exchange and partial command request. |
| **Parameters** | `svGetRequest` — the **SV Get** ingoing command data.<br>`svGetResponse` — the **SV Get** outgoing command data.<br>`svCommandPartialRequest` — the partial **SV Load/Debit/Undebit** ingoing command data, to which the computed security data is later appended. |
| **Returns** | A `SvCommandSecurityData` carrying the security data to place in the **SV Load/Debit/Undebit** command request. |
| **Pre-conditions** | **Argument** — every parameter is non-`null`.<br>**Argument** — every parameter is non-empty. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Deactivate Encryption*

| | |
|---|---|
| **Signature** | `deactivateEncryption() → Unit` |
| **Since** | 0.1 |
| **Description** | Deactivates the encryption/decryption of the data sent/received during the secure session. |
| **Pre-conditions** | None. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Finalize Terminal Session MAC*

| | |
|---|---|
| **Signature** | `finalizeTerminalSessionMac() → ByteArray` |
| **Since** | 0.1 |
| **Description** | Finalises the digest computation and returns the terminal part of the session MAC. |
| **Returns** | A non-empty `ByteArray` containing the terminal session MAC. |
| **Pre-conditions** | None. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Generate Ciphered Card Key*

| | |
|---|---|
| **Signature** | `generateCipheredCardKey(`<br>`  cardChallenge: ByteArray,`<br>`  issuerKeyKif: Byte,`<br>`  issuerKeyKvc: Byte,`<br>`  targetKeyKif: Byte,`<br>`  targetKeyKvc: Byte`<br>`) → ByteArray` |
| **Since** | 0.1 |
| **Description** | Generates an encrypted key data block for loading a key into a card. |
| **Parameters** | `cardChallenge` — a `ByteArray` containing the card challenge.<br>`issuerKeyKif` — the issuer key KIF.<br>`issuerKeyKvc` — the issuer key KVC.<br>`targetKeyKif` — the target key KIF.<br>`targetKeyKvc` — the target key KVC. |
| **Returns** | A non-empty `ByteArray` containing the encrypted data block to send to the card. |
| **Pre-conditions** | **Argument** — every parameter is non-`null`. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Generate Terminal Session MAC*

| | |
|---|---|
| **Signature** | `generateTerminalSessionMac() → ByteArray` |
| **Since** | 0.1 |
| **Description** | Generates the terminal part of the session MAC used for an **early mutual authentication**. |
| **Returns** | A non-empty `ByteArray` containing the terminal session MAC. |
| **Pre-conditions** | None. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Init Terminal Secure Session Context*

| | |
|---|---|
| **Signature** | `initTerminalSecureSessionContext() → ByteArray` |
| **Since** | 0.1 |
| **Description** | Initialises the crypto module context for operating a **Secure Session** with a card and returns the terminal challenge. |
| **Returns** | The non-empty terminal challenge. |

| | |
|---|---|
| **Pre-conditions** | None. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Init Terminal Session MAC*

| | |
|---|---|
| **Signature** | `initTerminalSessionMac(`<br>`  openSecureSessionDataOut: ByteArray,`<br>`  kif: Byte,`<br>`  kvc: Byte`<br>`) → Unit` |
| **Since** | 0.1 |
| **Description** | Stores the data needed to initialise the session MAC computation for a **Secure Session**. |
| **Parameters** | `openSecureSessionDataOut` — the data out from the card **Open Secure Session** command.<br>`kif` — the card KIF.<br>`kvc` — the card KVC. |
| **Pre-conditions** | **Argument** — every parameter is non-`null`. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Is Card Session MAC Valid*

| | |
|---|---|
| **Signature** | `isCardSessionMacValid(cardSessionMac: ByteArray) → Boolean` |
| **Since** | 0.1 |
| **Description** | Verifies the card part of the session MAC, finalising the mutual authentication process. |
| **Parameters** | `cardSessionMac` — a `ByteArray` containing the card session MAC. |
| **Returns** | `true` if the card session MAC is validated, `false` otherwise. |
| **Pre-conditions** | **Argument** — every parameter is non-`null`. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Is Card SV MAC Valid*

| | |
|---|---|
| **Signature** | `isCardSvMacValid(cardSvMac: ByteArray) → Boolean` |
| **Since** | 0.1 |
| **Description** | Verifies the Stored Value (SV) MAC returned by the card, confirming the authenticity of the SV operation. |
| **Parameters** | `cardSvMac` — a `ByteArray` containing the card SV MAC. |
| **Returns** | `true` if the card SV MAC is validated, `false` otherwise. |
| **Pre-conditions** | **Argument** — every parameter is non-`null`. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

### *Synchronize*

| | |
|---|---|
| **Signature** | `synchronize() → Unit` |
| **Since** | 0.1 |

| | |
|---|---|
| **Description** | Synchronises data of the associated card transaction crypto extension if needed. |
| **Pre-conditions** | None. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

#### *Update Terminal Session MAC*

| | |
|---|---|
| **Signature** | `updateTerminalSessionMac(cardApdu: ByteArray) → ByteArray?` |
| **Since** | 0.1 |
| **Description** | Updates the digest computation with data sent or received from the card. Returns the encrypted/decrypted data when the encryption is active. The digest is computed **incrementally** and depends on the sequence of inputs: the implementation MUST preserve the order in which this operation is called. |
| **Parameters** | `cardApdu` — a `ByteArray` containing either the input or output data of a card command APDU. |
| **Returns** | `null` if the encryption is not active; the non-empty ciphered or deciphered command data otherwise. |
| **Pre-conditions** | **Argument** — every parameter is non-`null`. |
| **Errors** | `SymmetricCrypto` — if an internal error occurred.<br>`SymmetricCryptoIO` — if an IO error occurred when processing a command. |

## 5.4. Errors

Every error defined below carries two properties: `message` (`String`), which describes the condition, and `cause` (`Any?`), the underlying error that triggered it, `null` when there is none. They are not restated for each error.

### *5.4.1. Symmetric Crypto*

| | |
|---|---|
| **Name** | `SymmetricCrypto` |
| **Kind** | Error |
| **Namespace** | `calypso.crypto.symmetric` |
| **Since** | 0.1 |
| **Purpose** | Indicates that an internal error occurred when processing a command. |

### *5.4.2. Symmetric Crypto IO*

| | |
|---|---|
| **Name** | `SymmetricCryptoIO` |
| **Kind** | Error |
| **Namespace** | `calypso.crypto.symmetric` |
| **Since** | 0.1 |
| **Purpose** | Indicates that an IO error occurred when processing a command. |