SPECIFICATION

Calypso Terminal API

Calypso Crypto Symmetric

Version 0.1.2-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 . . . 5
2. Introduction . . . 6
2.1. Purpose . . . 6
2.2. Scope . . . 6
2.3. Conformance . . . 6
2.4. Document Conventions . . . 7
2.4.1. Naming . . . 7
2.4.2. Language-agnostic types . . . 7
2.4.3. Type tables . . . 7
2.4.4. Operation tables . . . 7
2.4.5. Operation contracts . . . 8
2.4.6. Concurrency . . . 8
2.5. References and Resources . . . 8
2.5.1. Calypso References . . . 8
2.5.2. Normative References . . . 8
2.5.3. External Resources . . . 8
3. Glossary and Acronyms . . . 9
3.1. Glossary . . . 9
3.2. Acronyms . . . 9
4. Architectural Overview . . . 10
4.1. Functional positioning . . . 10
4.2. Logical namespaces . . . 10
4.3. UML class diagram . . . 10
5. API Specification . . . 12
5.1. Classes . . . 12
5.1.1. Symmetric Crypto API Properties . . . 12
Constants . . . 12
5.2. API Interfaces . . . 12
5.2.1. SV Command Security Data API . . . 12
Get SV Command Partial Request . . . 12
Get SV Get Request . . . 12
Get SV Get Response . . . 13
Set Serial Number . . . 13
Set Terminal Challenge . . . 13
Set Terminal SV MAC . . . 13
Set Transaction Number . . . 13
5.3. SPI Interfaces . . . 14
5.3.1. Symmetric Crypto Card Transaction Manager Factory SPI . . . 14
Create Card Transaction Manager . . . 14
Get Max Card APDU Length Supported . . . 14
Is Extended Mode Supported . . . 14
Pre Init Terminal Session Context . . . 15
5.3.2. Symmetric Crypto Card Transaction Manager SPI . . . 15
Activate Encryption . . . 15
Cipher PIN For Modification . . . 15
Cipher PIN For Presentation . . . 16
Compute SV Command Security Data . . . 16

Deactivate Encryption . . . 16
Finalize Terminal Session MAC . . . 17
Generate Ciphered Card Key . . . 17
Generate Terminal Session MAC . . . 17
Init Terminal Secure Session Context . . . 17
Init Terminal Session MAC . . . 18
Is Card Session MAC Valid . . . 18
Is Card SV MAC Valid . . . 18
Synchronize . . . 18
Update Terminal Session MAC . . . 18
5.4. Exceptions . . . 19
5.4.1. Symmetric Crypto Exception . . . 19
5.4.2. Symmetric Crypto IO Exception . . . 19

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, the interfaces, classes, enumerations, exceptions, 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	0.1.2-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 0.1.2-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
v0.1.2-SNAPSHOT 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 container used to carry SV security data,
- the exceptions 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 exceptions exclusively of the types defined in Section 5.4 for the situations described,
3. it preserves the structural relationships described in Chapter 4.

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. Language-agnostic types

Symbolic 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`.
`integer`	Signed 32-bit integer.	`int`, `Int32`.
`byte array`	Ordered sequence of octets.	`byte[]`, `[u8]`, `bytes`.
`list<T>`	Ordered collection of `T` values, may contain duplicates.	`List<T>`, `Vec<T>`.
`void`	Indicates that an operation returns no value.	`void`, `Unit`, `None`.

2.4.3. Type tables

Each interface, class, enumeration and exception 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, exception, 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 language-agnostic 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.

Throws	A list of exceptional conditions, each mapped to a specific exception 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:

- **Non-`null` arguments.** Unless explicitly stated otherwise, every input parameter MUST be non-`null`. An implementation MUST raise `IllegalArgumentException` if a `null` value is supplied. This implicit `null` check is not repeated in the **Throws** row, which states `None.` when no other exception can occur.
- **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

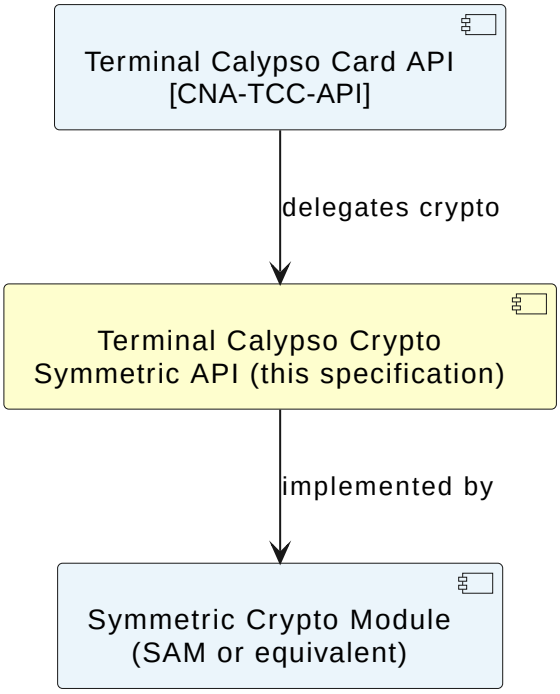

Figure 1. Terminal Calypso Crypto Symmetric API — Architecture Overview

4.2. Logical namespaces

Namespace	Role	Summary
`calypso.crypto.symmetric`	Public API	Properties, data carrier and exceptions consumed by both sides of the SPI.
`calypso.crypto.symmetric.spi`	SPI	Interfaces implemented by the symmetric crypto module.

4.3. UML class diagram

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

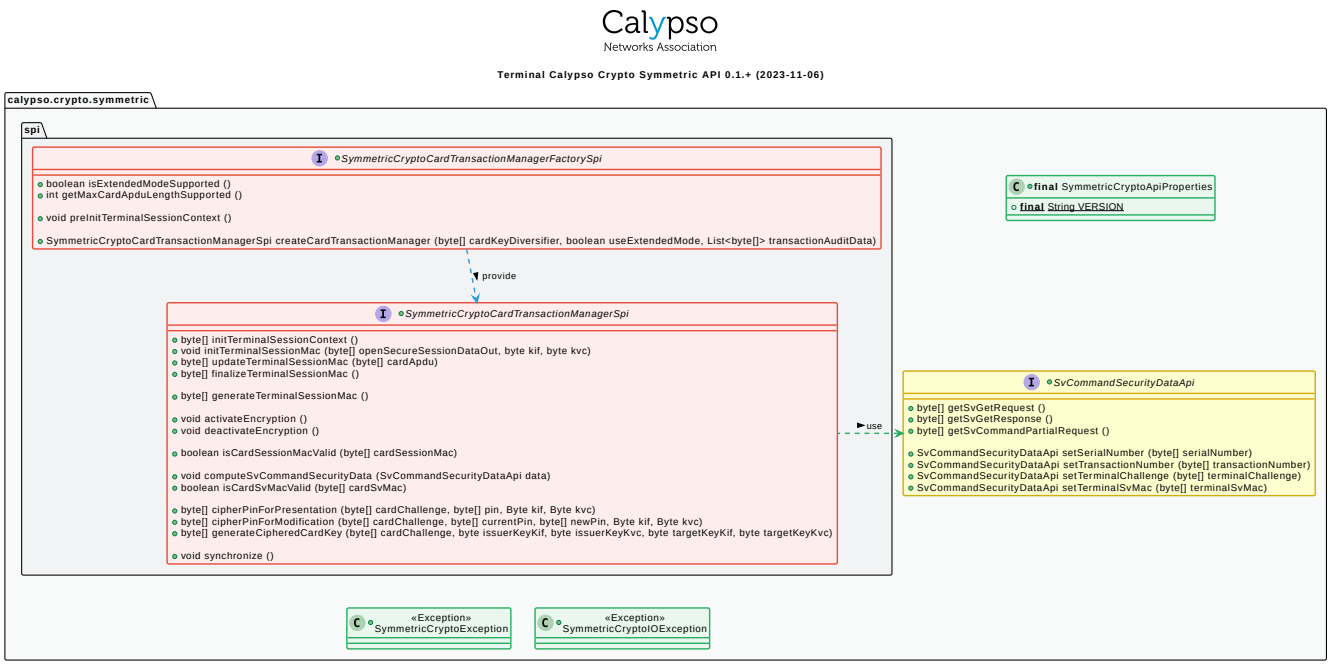

Figure 2. Terminal Calypso Crypto Symmetric API — UML Class Diagram

5. API Specification

5.1. Classes

5.1.1. Symmetric Crypto API Properties

Name	`SymmetricCryptoApiProperties`
Kind	Final class
Namespace	`calypso.crypto.symmetric`
Since	0.1
Purpose	Exposes the immutable properties of the Terminal Calypso Crypto Symmetric API.

Constants

Name	Type	Since	Description
`VERSION`	`string`	0.1	String representation of the version of the API implemented by the conforming binding (e.g. `"0.1"`). The value is a dotted decimal of the form `MAJOR.MINOR`.

The `SymmetricCryptoApiProperties` type cannot be instantiated; it exposes only static properties.

5.2. API Interfaces

5.2.1. SV Command Security Data API

Name	`SvCommandSecurityDataApi`
Kind	Interface
Namespace	`calypso.crypto.symmetric`
Since	0.1
Purpose	Carries the input/output data exchanged during the cryptographic preparation of an SV command (Load / Debit / Undebit).

Get SV Command Partial Request

Signature	`getSvCommandPartialRequest() → byte array`
Since	0.1
Description	Returns the partial **SV Load/Debit/Undebit** ingoing command data, to which the security data computed by the crypto module is later appended.
Returns	A non-empty `byte array` containing the **SV Load/Debit/Undebit** APDU partial request.
Throws	None.

Get SV Get Request

Signature	`getSvGetRequest() → byte array`

Since	0.1
Description	Returns the **SV Get** ingoing command data.
Returns	A non-empty `byte array` containing the **SV Get** APDU request data.
Throws	None.

Get SV Get Response

Signature	`getSvGetResponse() → byte array`
Since	0.1
Description	Returns the **SV Get** outgoing command data.
Returns	A non-empty `byte array` containing the **SV Get** APDU response data.
Throws	None.

Set Serial Number

Signature	`setSerialNumber(serialNumber: byte array) → SvCommandSecurityDataApi`
Since	0.1
Description	Sets the serial number to be placed in the **SV Load/Debit/Undebit** command request.
Parameters	`serialNumber` — the serial number to be used.
Returns	The current instance (`SvCommandSecurityDataApi`).
Throws	None.

Set Terminal Challenge

Signature	`setTerminalChallenge(terminalChallenge: byte array) → SvCommandSecurityDataApi`
Since	0.1
Description	Sets the terminal challenge to be placed in the **SV Load/Debit/Undebit** command request.
Parameters	`terminalChallenge` — the terminal challenge to be used.
Returns	The current instance (`SvCommandSecurityDataApi`).
Throws	None.

Set Terminal SV MAC

Signature	`setTerminalSvMac(terminalSvMac: byte array) → SvCommandSecurityDataApi`
Since	0.1
Description	Sets the terminal SV MAC to be placed in the **SV Load/Debit/Undebit** command request.
Parameters	`terminalSvMac` — the terminal SV MAC to be used.
Returns	The current instance (`SvCommandSecurityDataApi`).
Throws	None.

Set Transaction Number

Signature	`setTransactionNumber(transactionNumber: byte array) → SvCommandSecurityDataApi`
Since	0.1
Description	Sets the transaction number to be placed in the **SV Load/Debit/Undebit** command request.

Parameters	`transactionNumber` — the transaction number to be used.
Returns	The current instance (`SvCommandSecurityDataApi`).
Throws	None.

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(`
	`  cardKeyDiversifier: byte array,`
	`  useExtendedMode: boolean,`
	`  transactionAuditData: list<byte array>`
	`) → 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. `useExtendedMode` — a flag requesting the extended mode, if supported by the crypto service. `transactionAuditData` — the reference of the list where the transaction audit data are recorded.
Returns	A `SymmetricCryptoCardTransactionManagerSpi`.
Throws	`IllegalStateException` — if the extended mode is requested but not supported.

Get Max Card APDU Length Supported

Signature	`getMaxCardApduLengthSupported() → integer`
Since	0.1
Description	Returns the maximum card APDU length supported by the crypto module.
Returns	A positive `integer`.
Throws	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.
Throws	None.

Pre Init Terminal Session Context

Signature	`preInitTerminalSessionContext() → void`
Since	0.1
Description	Retrieves and stores the terminal challenge in the crypto module image for later use.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — 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 `SymmetricCryptoException` (internal error) or `SymmetricCryptoIOException` (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() → void`
Since	0.1
Description	Activates the encryption/decryption of the data sent/received during the secure session.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Cipher PIN For Modification

Signature	`cipherPinForModification(`
	`  cardChallenge: byte array,`
	`  currentPin: byte array,`
	`  newPin: byte array,`
	`  kif: byte,`
	`  kvc: byte`
	`) → byte array`

Since	0.1
Description	Computes a block of encrypted data to be sent to the card for a PIN modification. **Note:** the `kif` and `kvc` parameters MUST be ignored when PIN modification is performed within a Secure Session.

Parameters	`cardChallenge` — a `byte array` containing the card challenge. `currentPin` — a `byte array` containing the 4-byte current PIN value. `newPin` — a `byte array` containing the 4-byte new PIN value. `kif` — the PIN encryption key KIF. `kvc` — the PIN encryption key KVC.
Returns	A non-empty `byte array` containing the encrypted data block to send to the card.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Cipher PIN For Presentation

Signature	`cipherPinForPresentation(`
	`  cardChallenge: byte array,`
	`  pin: byte array,`
	`  kif: byte,`
	`  kvc: byte`
	`) → byte array`

Since	0.1
Description	Computes a block of encrypted data to be sent to the card for an enciphered PIN presentation. **Note:** the `kif` and `kvc` parameters MUST be ignored when PIN verification is performed within a Secure Session.
Parameters	`cardChallenge` — a `byte array` containing the card challenge. `pin` — a `byte array` containing the 4-byte PIN value. `kif` — the PIN encryption key KIF. `kvc` — the PIN encryption key KVC.
Returns	A non-empty `byte array` containing the encrypted data block to send to the card.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Compute SV Command Security Data

Signature	`computeSvCommandSecurityData(data: SvCommandSecurityDataApi) → void`
Since	0.1
Description	Computes the data needed to operate SV card commands and writes it back to the provided container. Every field of the supplied `SvCommandSecurityDataApi` required to assemble the complete SV command APDU MUST be populated.
Parameters	`data` (`SvCommandSecurityDataApi`) — the data involved in the preparation of an SV Reload/Debit/Undebit command.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Deactivate Encryption

Signature	`deactivateEncryption() → void`
Since	0.1
Description	Deactivates the encryption/decryption of the data sent/received during the secure session.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Finalize Terminal Session MAC

Signature	`finalizeTerminalSessionMac() → byte array`
Since	0.1
Description	Finalises the digest computation and returns the terminal part of the session MAC.
Returns	A non-empty `byte array` containing the terminal session MAC.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Generate Ciphered Card Key

Signature	`generateCipheredCardKey(`
	`  cardChallenge: byte array,`
	`  issuerKeyKif: byte,`
	`  issuerKeyKvc: byte,`
	`  targetKeyKif: byte,`
	`  targetKeyKvc: byte`
	`) → byte array`

Since	0.1
Description	Generates an encrypted key data block for loading a key into a card.
Parameters	`cardChallenge` — a `byte array` containing the card challenge. `issuerKeyKif` — the issuer key KIF. `issuerKeyKvc` — the issuer key KVC. `targetKeyKif` — the target key KIF. `targetKeyKvc` — the target key KVC.
Returns	A non-empty `byte array` containing the encrypted data block to send to the card.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Generate Terminal Session MAC

Signature	`generateTerminalSessionMac() → byte array`
Since	0.1
Description	Generates the terminal part of the session MAC used for an **early mutual authentication**.
Returns	A non-empty `byte array` containing the terminal session MAC.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Init Terminal Secure Session Context

Signature	`initTerminalSecureSessionContext() → byte array`
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.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Init Terminal Session MAC

Signature	`initTerminalSessionMac(`
	`  openSecureSessionDataOut: byte array,`
	`  kif: byte,`
	`  kvc: byte`
	`) → void`

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. `kif` — the card KIF. `kvc` — the card KVC.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Is Card Session MAC Valid

Signature	`isCardSessionMacValid(cardSessionMac: byte array) → boolean`
Since	0.1
Description	Verifies the card part of the session MAC, finalising the mutual authentication process.
Parameters	`cardSessionMac` — a `byte array` containing the card session MAC.
Returns	`true` if the card session MAC is validated, `false` otherwise.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Is Card SV MAC Valid

Signature	`isCardSvMacValid(cardSvMac: byte array) → 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 `byte array` containing the card SV MAC.
Returns	`true` if the card SV MAC is validated, `false` otherwise.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Synchronize

Signature	`synchronize() → void`
Since	0.1
Description	Synchronises data of the associated card transaction crypto extension if needed.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

Update Terminal Session MAC

Signature	`updateTerminalSessionMac(cardApdu: byte array) → byte array?`
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 `byte array` 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.
Throws	`SymmetricCryptoException` — if an internal error occurred. `SymmetricCryptoIOException` — if an IO error occurred when processing a command.

5.4. Exceptions

5.4.1. Symmetric Crypto Exception

Name	`SymmetricCryptoException`
Kind	Checked exception
Namespace	`calypso.crypto.symmetric`
Since	0.1
Purpose	Indicates that an internal error occurred when processing a command.

5.4.2. Symmetric Crypto IO Exception

Name	`SymmetricCryptoIOException`
Kind	Checked exception
Namespace	`calypso.crypto.symmetric`
Since	0.1
Purpose	Indicates that an IO error occurred when processing a command.