SPECIFICATION

Calypso Terminal API

Calypso Crypto Asymmetric

Version 0.2.1-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 Asymmetric 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 Asymmetric API MUST satisfy.

The Terminal Calypso Crypto Asymmetric API is the SPI through which the **Terminal Calypso Card API** delegates every asymmetric-key cryptographic operation — in particular PKI-mode secure session computations and the validation of the certification chain associated with PKI-enabled Calypso cards.

Document Status

Reference	YYMMDD-SP-CNATerminalAPI-CalypsoCryptoAsymmetric
Short name	CNA-TCCA-API
Version	0.2.1-SNAPSHOT
Revision date	2026-07-20
Editor	Calypso Networks Association
Source repository	https://github.com/calypsonet/calypsonet-terminal-calypso-crypto-asymmetric-uml-api
Reference license	Creative Commons Attribution-NoDerivatives 4.0 International (CC BY-ND 4.0)

The present document specifies the 0.2.1-SNAPSHOT of the Terminal Calypso Crypto Asymmetric 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
v0.2.1-SNAPSHOT 2026-07-20	Baseline.

2. Introduction

2.1. Purpose

The Terminal Calypso Crypto Asymmetric API defines the contract that an **asymmetric-key crypto module** MUST expose so that a Calypso transaction manager can:

- parse, validate and chain Primary CA (PCA), CA and card certificates produced by the Calypso PKI;
- operate a PKI-mode secure session against a Calypso card (init, update, verify card signature).

The contract is intentionally agnostic to the underlying cryptographic library: any component capable of producing the required signatures (as specified in ISO/IEC 9796) and certificate validations MUST be acceptable.

2.2. Scope

This specification covers:

- the data carriers used to expose certificate content and public keys to the upstream layer,
- the SPIs implemented by the crypto module to parse and validate certificates,
- the SPI implemented by the crypto module to drive a PKI secure session,
- the factory used by the upstream layer to instantiate the transaction-manager SPI,
- the exceptions raised by the SPI.

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

- the on-the-wire format of certificates beyond their parsing entry point,
- the management of certificate stores and revocation,
- the cryptographic algorithms themselves.

2.3. Conformance

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

1. it implements every interface defined in Chapter 5 with the operations and semantics prescribed in this document,
2. it raises exceptions exclusively of the types defined in Section 5.3 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` except for enumeration values which use `UPPER_SNAKE_CASE`. Names defined by this specification are reproduced as-is in the UML class diagram (Section 4.3).

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`.
`long`	Signed 64-bit integer.	`long`, `Int64`.
`byte array`	Ordered sequence of octets.	`byte[]`, `[u8]`, `bytes`.
`publicKey`	Opaque, immutable handle representing an asymmetric public key.	`PublicKey` (Java), `X509PublicKey`, equivalent.
`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.

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.
[ISO/IEC 9796]	*Information technology — Security techniques — Digital signature schemes giving message recovery*.

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
PCA	Primary Certification Authority — the trust anchor of the Calypso PKI certificate hierarchy. Self-signed.
CA	Certification Authority — intermediate authority signed by the PCA or another CA, used to authenticate further CAs or cards.
Card Certificate	Certificate signed by a CA, binding a public key to the card identity.
PKI Mode	Calypso secure-session mode in which authentication of the card relies on asymmetric cryptography.
Public Key Reference	Stable identifier (typically the public key hash) used to look up the issuer's public key.
SPI	Service Provider Interface — an interface designed to be implemented by an extension module.

3.2. Acronyms

Abbreviation	Expansion
AID	Application IDentifier
APDU	Application Protocol Data Unit
CA	Certification Authority
CNA	Calypso Networks Association
PCA	Primary Certification Authority
PKI	Public Key Infrastructure
SPI	Service Provider Interface
UML	Unified Modelling Language

4. Architectural Overview

4.1. Functional positioning

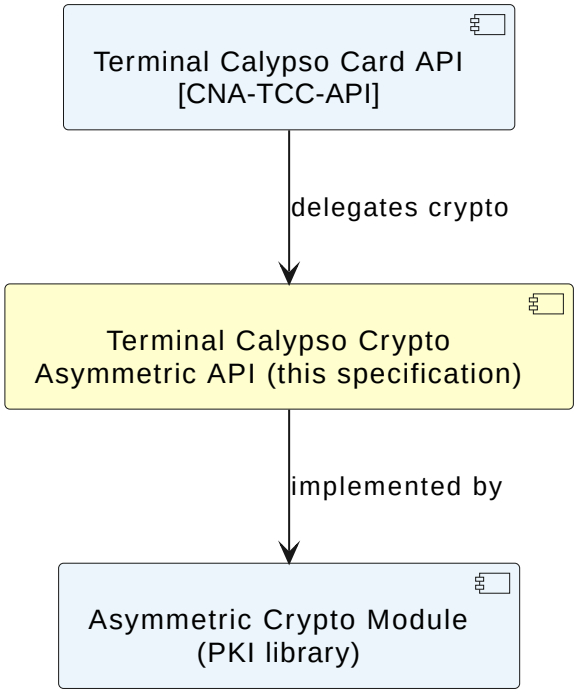

Figure 1. Terminal Calypso Crypto Asymmetric API — Architecture Overview

4.2. Logical namespaces

Namespace	Role	Summary
`calypso.crypto.asymmetric`	Public API	Properties and root exceptions.
`calypso.crypto.asymmetric.certificate`	Public API	Certificate-validation exception type.
`calypso.crypto.asymmetric.certificate.spi`	SPI	Interfaces implemented by the crypto module for certificate parsing, validation and public-key exposure.
`calypso.crypto.asymmetric.transaction.spi`	SPI	Interfaces implemented by the crypto module to drive a PKI-mode card transaction.

4.3. UML class diagram

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

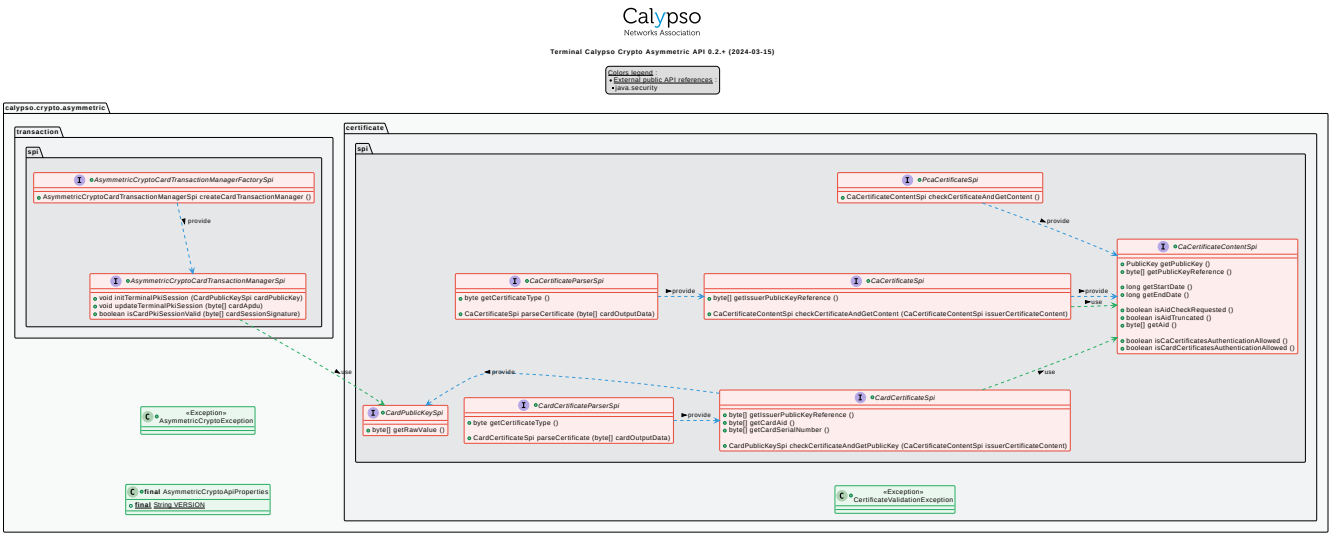

Figure 2. Terminal Calypso Crypto Asymmetric API — UML Class Diagram

5. API Specification

5.1. Classes

5.1.1. Asymmetric Crypto API Properties

Name	`AsymmetricCryptoApiProperties`
Kind	Final class
Namespace	`calypso.crypto.asymmetric`
Since	0.2
Purpose	Exposes the immutable properties of the Terminal Calypso Crypto Asymmetric API.

Constants

Name	Type	Since	Description
`VERSION`	`string`	0.2	String representation of the version of the API implemented by the conforming binding (e.g. `"0.2"`).

5.2. SPI Interfaces

Certificate validation follows a **chain**: it MUST start from a PCA certificate (validated through `PcaCertificateSpi.checkCertificateAndGetContent`), traverse zero or more CA certificates, and end on a card certificate that exposes the card's public key. Each `checkCertificateAndGetContent` / `checkCertificateAndGetPublicKey` call MUST validate the signature **and** the metadata (validity dates, AID constraints, certificate purpose) before returning a value.

5.2.1. Asymmetric Crypto Card Transaction Manager Factory SPI

Name	`AsymmetricCryptoCardTransactionManagerFactorySpi`
Kind	Interface (SPI)
Namespace	`calypso.crypto.asymmetric.transaction.spi`
Since	0.2
Purpose	Factory of `AsymmetricCryptoCardTransactionManagerSpi`.

Create Card Transaction Manager

Signature	`createCardTransactionManager() → AsymmetricCryptoCardTransactionManagerSpi`
Since	0.2
Description	Creates an `AsymmetricCryptoCardTransactionManagerSpi` providing the cryptographic primitives that a Calypso card transaction requires when using asymmetric (PKI) keys. Implemented by the crypto module.
Returns	An `AsymmetricCryptoCardTransactionManagerSpi`.
Throws	None.

5.2.2. Asymmetric Crypto Card Transaction Manager SPI

Name	`AsymmetricCryptoCardTransactionManagerSpi`
Kind	Interface (SPI)
Namespace	`calypso.crypto.asymmetric.transaction.spi`
Since	0.2
Purpose	Defines the cryptographic primitives required by a Calypso card transaction when using asymmetric keys. An instance is obtained via `AsymmetricCryptoCardTransactionManagerFactorySpi.createCardTransactionManager`.

Init Terminal PKI Session

Signature	`initTerminalPkiSession(cardPublicKey: CardPublicKeySpi) → void`
Since	0.2
Description	Initialises the cryptographic context for a new PKI secure session with the supplied card public key.
Parameters	`cardPublicKey` (`CardPublicKeySpi`) — the card public key, extracted from the previously validated card certificate.
Throws	`AsymmetricCryptoException` — if the provided public key value is not compliant with the current elliptic curve, or if an error occurs during the initialisation.

Is Card PKI Session Valid

Signature	`isCardPkiSessionValid(cardSessionSignature: byte array) → boolean`
Since	0.2
Description	Verifies the provided secure session signature against the previously initialised and updated context. This MUST be the final step of the PKI secure session, called after the last `updateTerminalPkiSession`.
Parameters	`cardSessionSignature` — the 64-byte card signature to verify.
Returns	`true` if the signature is valid, `false` otherwise.
Throws	`AsymmetricCryptoException` — if an error occurs while verifying the signature.

Update Terminal PKI Session

Signature	`updateTerminalPkiSession(cardApdu: byte array) → void`
Since	0.2
Description	Updates the session signature verification engine with data sent to or received from the card. This operation MUST be called for every APDU exchanged during the PKI secure session, in the order in which it was exchanged. • For ingoing data, the input length MUST be `>= 5`. • For outgoing data, the input length MUST be `>= 2`.
Parameters	`cardApdu` — the APDU bytes exchanged with the card (ingoing or outgoing).
Throws	`AsymmetricCryptoException` — if an error occurs while updating the session.

5.2.3. CA Certificate Content SPI

Name	`CaCertificateContentSpi`
Kind	Interface (SPI)

Namespace	`calypso.crypto.asymmetric.certificate.spi`
Since	0.2
Purpose	Exposes the content of a CA certificate that has been parsed and verified by the crypto module.

Get AID

Signature	`getAid() → byte array?`
Since	0.2
Description	Returns the AID value carried by the certificate.
Returns	The non-empty AID, or `null` if the AID check is not requested.
Throws	None.

Get End Date

Signature	`getEndDate() → long`
Since	0.2
Description	Returns the validity end date of the certificate as a `long` formatted `0xYYYYMMDD`. Returns `0` if the validity end date is not defined or available.
Returns	The end date, or `0` when not defined.
Throws	None.

Get Public Key

Signature	`getPublicKey() → publicKey`
Since	0.2
Description	Returns the public key carried by the certificate.
Returns	A public-key handle.
Throws	None.

Get Public Key Reference

Signature	`getPublicKeyReference() → byte array`
Since	0.2
Description	Returns the reference of the public key as a `byte array`. The reference is the stable identifier used by other certificates to designate this key as their issuer.
Returns	A non-empty `byte array`.
Throws	None.

Get Start Date

Signature	`getStartDate() → long`
Since	0.2
Description	Returns the validity start date of the certificate as a `long` formatted `0xYYYYMMDD` (four-digit year, two-digit month, two-digit day). Returns `0` if the validity start date is not defined or available.
Returns	The start date, or `0` when not defined.
Throws	None.

Is AID Check Requested

Signature	`isAidCheckRequested() → boolean`
Since	0.2
Description	Indicates whether the AID associated with the certificate MUST be checked.
Returns	`true` if the AID has to be checked, `false` otherwise.
Throws	None.

Is AID Truncated

Signature	`isAidTruncated() → boolean`
Since	0.2
Description	Indicates whether the AID carried by the certificate is truncated.
Returns	`true` if the AID is truncated, `false` otherwise.
Throws	None.

Is CA Certificates Authentication Allowed

Signature	`isCaCertificatesAuthenticationAllowed() → boolean`
Since	0.2
Description	Indicates whether the certificate is allowed to authenticate further CA certificates.
Returns	`true` if CA-certificate authentication is allowed, `false` otherwise.
Throws	None.

Is Card Certificates Authentication Allowed

Signature	`isCardCertificatesAuthenticationAllowed() → boolean`
Since	0.2
Description	Indicates whether the certificate is allowed to authenticate card certificates.
Returns	`true` if card-certificate authentication is allowed, `false` otherwise.
Throws	None.

5.2.4. CA Certificate Parser SPI

Name	`CaCertificateParserSpi`
Kind	Interface (SPI)
Namespace	`calypso.crypto.asymmetric.certificate.spi`
Since	0.2
Purpose	Parses CA certificates from raw data stored on a card.

Get Certificate Type

Signature	`getCertificateType() → byte`
Since	0.2
Description	Returns the certificate type identifier associated with the parser.

Returns	A `byte` value.
Throws	None.

Parse Certificate

Signature	`parseCertificate(cardOutputData: byte array) → CaCertificateSpi`
Since	0.2
Description	Parses the supplied card output data and returns a new CA certificate instance. The first byte of the input array MUST be the certificate type.
Parameters	`cardOutputData` — a `byte array` containing the CA certificate as stored on the card.
Returns	A `CaCertificateSpi`.
Throws	`CertificateValidationException` — if the provided certificate has an unsupported format.

5.2.5. CA Certificate SPI

Name	`CaCertificateSpi`
Kind	Interface (SPI)
Namespace	`calypso.crypto.asymmetric.certificate.spi`
Since	0.2
Purpose	Represents a Certification Authority (CA) certificate that has been parsed by the crypto module and is ready for chain validation.

Check Certificate And Get Content

Signature	`checkCertificateAndGetContent(`
	`  issuerCertificateContent: CaCertificateContentSpi`
	`) → CaCertificateContentSpi`

Since	0.2
Description	Verifies the certificate signature and other relevant fields, then returns the certificate content. The validation MUST be comprehensive: signature correctness, validity period, issuer and subject details and any relevant constraints or extensions.
Parameters	`issuerCertificateContent` (`CaCertificateContentSpi`) — the issuer certificate content to be used for signature verification.
Returns	A `CaCertificateContentSpi`.
Throws	`CertificateValidationException` — if the certificate is invalid, expired, revoked, or fails any other validation. `AsymmetricCryptoException` — if a technical error occurs during the cryptographic computations.

Get Issuer Public Key Reference

Signature	`getIssuerPublicKeyReference() → byte array`
Since	0.2
Description	Returns the reference of the issuer's public key, used by the upstream layer to locate the corresponding issuer certificate.
Returns	A non-empty `byte array`.
Throws	None.

5.2.6. Card Certificate Parser SPI

Name	`CardCertificateParserSpi`
Kind	Interface (SPI)
Namespace	`calypso.crypto.asymmetric.certificate.spi`
Since	0.2
Purpose	Parses card certificates from raw data stored on a card.

Get Certificate Type

Signature	`getCertificateType() → byte`
Since	0.2
Description	Returns the certificate type identifier associated with the parser.
Returns	A `byte` value.
Throws	None.

Parse Certificate

Signature	`parseCertificate(cardOutputData: byte array) → CardCertificateSpi`
Since	0.2
Description	Parses the supplied card output data and returns a new card certificate instance. The first byte of the input MUST be the certificate type. The expected total length is 316 bytes.
Parameters	`cardOutputData` — a `byte array` containing the card certificate as stored on the card (316 bytes).
Returns	A `CardCertificateSpi`.
Throws	`CertificateValidationException` — if the provided certificate has an unsupported format.

5.2.7. Card Certificate SPI

Name	`CardCertificateSpi`
Kind	Interface (SPI)
Namespace	`calypso.crypto.asymmetric.certificate.spi`
Since	0.2
Purpose	Represents a card certificate, used to extract the card's public key after a successful chain validation.

Check Certificate And Get Public Key

Signature	`checkCertificateAndGetPublicKey(`
	`  issuerCertificateContent: CaCertificateContentSpi`
	`) → CardPublicKeySpi`

Since	0.2
Description	Verifies the certificate signature and other relevant fields, then returns the card's public key.
Parameters	`issuerCertificateContent` (`CaCertificateContentSpi`) — the issuer certificate content to be used for signature verification.
Returns	A `CardPublicKeySpi`.
Throws	`CertificateValidationException` — if the certificate is invalid. `AsymmetricCryptoException` — if a technical error occurs during the cryptographic computations.

Get Card AID

Signature	`getCardAid() → byte array`
Since	0.2
Description	Returns the AID of the autonomous application of the card as a `byte array` of 5 to 16 bytes.
Returns	A non-empty `byte array` of 5 to 16 bytes.
Throws	None.

Get Card Serial Number

Signature	`getCardSerialNumber() → byte array`
Since	0.2
Description	Returns the serial number of the card as an 8-byte `byte array`.
Returns	An 8-byte `byte array`.
Throws	None.

Get Issuer Public Key Reference

Signature	`getIssuerPublicKeyReference() → byte array`
Since	0.2
Description	Returns the reference of the issuer's public key.
Returns	A non-empty `byte array`.
Throws	None.

5.2.8. Card Public Key SPI

Name	`CardPublicKeySpi`
Kind	Interface (SPI)
Namespace	`calypso.crypto.asymmetric.certificate.spi`
Since	0.2
Purpose	Exposes the card's public key extracted from a validated card certificate.

Get Raw Value

Signature	`getRawValue() → byte array`
Since	0.2
Description	Returns the raw value of the card's public key.
Returns	A 64-byte `byte array`.
Throws	None.

5.2.9. PCA Certificate SPI

Name	`PcaCertificateSpi`
Kind	Interface (SPI)
Namespace	`calypso.crypto.asymmetric.certificate.spi`

Since	0.2
Purpose	Represents a Primary Certification Authority (PCA) certificate—the self-signed trust anchor of the Calypso PKI.

Check Certificate And Get Content

Signature	`checkCertificateAndGetContent() → CaCertificateContentSpi`
Since	0.2
Description	Verifies the certificate signature and other relevant fields, then returns the certificate content. The PCA certificate MUST be expected to be self-signed in this context.
Returns	A `CaCertificateContentSpi`.
Throws	`CertificateValidationException`—if the certificate is invalid, expired, revoked, or fails any other validation. `AsymmetricCryptoException`—if a technical error occurs during the cryptographic computations.

5.3. Exceptions

5.3.1. Asymmetric Crypto Exception

Name	`AsymmetricCryptoException`
Kind	Checked exception
Namespace	`calypso.crypto.asymmetric`
Since	0.2
Purpose	Indicates that an error occurred when processing an asymmetric cryptographic operation.

5.3.2. Certificate Validation Exception

Name	`CertificateValidationException`
Kind	Checked exception
Namespace	`calypso.crypto.asymmetric.certificate`
Since	0.2
Purpose	Indicates an issue encountered during the certificate validation—e.g. an invalid signature or incorrect metadata values.