Revisions

|  |  |  |
| --- | --- | --- |
| **Rev** | **Date** | **Description** |
| 1.0 | 2018/09/13 | Initial reversion |
|  |  |  |
|  |  |  |
|  |  |  |

CATALOGUE

[coconut SDK manual for verifier 3](#_Toc528853584)

[1. Functions provided by the SDK 3](#_Toc528853585)

[2 SDK component release 3](#_Toc528853586)

[3 SDK integration method 3](#_Toc528853587)

[4 SDK API reference 3](#_Toc528853588)

[4.1 VerifySig method 3](#_Toc528853589)

[APPENDIX 5](#_Toc528853590)

[1. Error code list 5](#_Toc528853591)

[2. naming convention for the signature materials 6](#_Toc528853592)

[**Table 1 coconut SDK build&release details** 3](#_Toc528853593)

[**Table 2 naming convention details** 6](#_Toc528853594)

# coconut SDK manual for verifier

## Functions provided by the SDK

This SDK provide the related function of the Intel® EPID certificate signature verifying . Additionally, the SDK provide the interface to load test data, such as group public key, basename list, so that the SDK interfaces can be tested very conveniently.

## SDK component release

● For Linux

The SDK is internally implemented in C language, and the external interfaces are written in java native interface. the SDK component is released in the form of a dynamic link library(.so binary file) conforming to the JNI standard. After loading so file, java project can directly call the SDK interface of JNI form.

**Table 1 coconut SDK build&release details**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **OS** | **architecture** | **file type of SDK** | **file name of SDK** | **language (internal)** | **language (external)** | **compiler** |
| Linux | x86/x86\_64 | .so | libverifysig.so | C | JNI C++ | GCC tools chain |

## SDK integration method

● For Linux

In the java package of the web service, create the java package and the class with the same JNI interface as the java package and the class name, and then declare the native method of the JNI interface prototype released in the class; after the package and the class are created successfully, load the so file corresponds to the current architecture, and then these interfaces can be called in the java code according to the JNI interface prototype of the SDK.

## SDK API reference

### 4.1 VerifySig method

This interface is used to verify a signature signed by EPID certificate.

**Syntax**

|  |
| --- |
| int VerifySig(  String res\_directory\_path,  String sig\_file\_fullname,  String msg\_file\_fullname,  String basename  ) |

**Parameters**

|  |
| --- |
| res\_directory\_path |

the file path for storing the signature materials (for example, public key, encrypted private key, basename list, etc.)

|  |
| --- |
| sig\_file\_fullname |

the file path of the signature to be verified.

|  |
| --- |
| msg\_file\_fullname |

the file path of the message which was used to generate the signature to be verified.

|  |
| --- |
| basename |

the basename which was used to generate the signature to be verified.

**Return Value**

If executed successfully, it returns 0 (EPIDA\_OK), otherwise it returns the corresponding error code.

**Requirements(android)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Target Platform Linux

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DLL libverifysig.so

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Caller Language java

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## APPENDIX

### Error code list

|  |
| --- |
| typedef enum enCoconutSDKErrCode  {  EPIDA\_ERR = -1,  EPIDA\_OK = 0,  EPIDA\_INVALID\_PARAMETERS = 1,  EPIDA\_NO\_RES\_PATH = 2,  EPIDA\_INVALID\_RES\_PATH\_LEN = 3,  EPIDA\_NO\_CACER\_FILE = 4,  EPIDA\_INVALID\_CACER = 5,  EPIDA\_NO\_PUBKEY\_FILE = 6,  EPIDA\_INVALID\_PUBKEY = 7,  EPIDA\_NO\_PRIVKEY\_FILE = 8,  EPIDA\_INVALID\_PRIVKEY = 9,  EPIDA\_NO\_BSN\_LIST\_FILE = 10,  EPIDA\_INVALID\_BSN\_LIST = 11,  EPIDA\_SINMSG\_FAIL = 12,  EPIDA\_READ\_FILE\_FAIL = 13,  EPIDA\_WRITE\_FILE\_FAIL = 14,  EPIDA\_NO\_BSN\_SIG\_FILE\_PATH = 15,  EPIDA\_NO\_TRANS\_FILE\_PATH = 16,  EPIDA\_NO\_TRANS\_FILE = 17,  EPIDA\_NO\_TRANSC\_SIG\_PATH = 18,  EPIDA\_NO\_CREDENTIAL\_FILE\_PATH = 19,  EPIDA\_NO\_CREDENTIAL\_FILE = 20,  EPIDA\_INVALID\_CREDENTIAL = 21,  EPIDA\_NO\_MEMBERKEY\_FILE\_PATH = 22,  EPIDA\_NO\_MEMBERKEY\_FILE = 23,  EPIDA\_GEN\_MEMBERKEY\_FAIL = 24,  EPIDA\_NO\_SIG\_FILE\_PATH = 25,  EPIDA\_NO\_SIG\_FILE = 26,  EPIDA\_NO\_MSG\_FILE\_PATH = 27,  EPIDA\_NO\_MSG\_FILE = 28,  EPIDA\_NO\_VERIFY\_BASENAME = 29,  EPIDA\_SET\_VERIFIER\_BSN\_FAIL = 30,  EPIDA\_SET\_VERIFIER\_CTX\_FAIL = 31,  EPIDA\_VERIFY\_SIG\_FAIL = 32,  EPIDA\_NO\_NI\_FILE = 33,  EPIDA\_NO\_RANDOM\_FILE = 34,  EPIDA\_NO\_PRIVATEF\_FILE = 35,  EPIDA\_INVALID\_PRIVATEF = 36,  EPIDA\_NO\_JOINREQ\_FILE\_PATH = 37,  EPIDA\_MAKE\_JOINREQ\_FAIL = 38,  EPIDA\_NO\_PASSPHRASE = 39,  EPIDA\_ENCRYPT\_FAIL = 40,  EPIDA\_DECRYPT\_FAIL = 41,  }COCONUT\_ERR\_CODE\_E; |

### naming convention for the signature materials

The SDK uses some fixed file names internally. The caller must use the agreed file name.

**Table 2 naming convention details**

|  |  |  |
| --- | --- | --- |
| filename | file format | content |
| epid\_cacert.bin | binary | CA certificate |
| epid\_pubkey.bin | binary | EPID group public key |
| epid\_basenames.dat | JSON | basename list |