|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Document Version | Component Version | Date | Author | Department | Description | Status |
| v1.0 | V1.0 | 2021.01.11 | Andras Toth | XC-AN/ESW6-Bp | Software Developer | - |

|  |  |
| --- | --- |
| Keywords | CAPL, dll, ECDSA, SHA256 |
| Roles | Developer |
| Screens |  |
| Process steps |  |
| Document type |  |

|  |
| --- |
| General information  This manual is not suitable for double-sided printing.  This printout is not updated. The latest document version is published in the Bosch Learning space. |

**Table of Contents**

[1. Introduction 3](#_Toc61603910)

[2. SHA256 4](#_Toc61603911)

[3. ECDSA 5](#_Toc61603912)

[4. Possible return values 7](#_Toc61603913)

|  |
| --- |
| Introduction |
|  |

* The purpose of the documentation is to describe the functions, which are provided in the capldll.dll.
* The DLL was created in C++ using the CAPLdll example project provided by Vector.
  + C:\Users\Public\Documents\Vector\CANoe\Sample Configurations 11.0.81\Programming\CAPLdll
* For the cryptographic algorithms Microsoft’s Cryptography API: Next Generation (CNG) was used.
* CNG is validated to Federal Information Processing Standards (FIPS) 140-2 and is part of the Target of Evaluation for the Windows Common Criteria certification.
* For BLE/S32 chips the desired crypto algorithms are the following:
  + ECDSA – NISTP256 (Sign and Verify algorithm)
  + SHA256 (Hashing algorithm)

# SHA256

|  |  |
| --- | --- |
| Function name | dllSHA256\_Init() |
| Parameter | N/A |
| Return value | CNG status from C++ |

Description:

The function is responsible for initialising the SHA256 algorithm for usage.

If the return value is 0 (zero) the hashing algorithm was initialised correctly.

|  |  |
| --- | --- |
| Function name | dllSHA256\_Update(.., ..) |
| Byte Data[] | Data to be hashed |
| Long noOfBytes | No of bytes to be hashed |
| Return value | CNG status from C++ |

Description:

The function is responsible for updating a hash value. The function could be used for standalone hashing for fixed byte array or longer byte arrays, which could not be hashed in one step. (Array sizes are limited and working with them could be time consuming in CAPL)

If the update was correct 0 (zero) has to be returned.

|  |  |
| --- | --- |
| Function name | dllSHA256\_Finish(.., ..) |
| Byte Hash[] | Hashed message digest. |
| Long noOfBytes | SHA256 algorithm provides 32 bytes long hashes |
| Return value | CNG status from C++ |

Description:

After initialising and updating the required amount of data dllSHA256\_Finish() has to be called to provide the message digest (hash of the data).

Hashing the successfully finished if the return value is 0 (zero).

# ECDSA

|  |  |
| --- | --- |
| Function name | dllImport\_KeyPair |
| Byte KeyBlock[] – in | Key structure (104 bytes) |
| long noOfBytes | Keyblock size (104 fixed) |
| Return value | CNG status from C++ |

|  |  |
| --- | --- |
| KeyBlock structure | Size |
| BLOB | 8 bytes |
| Public Key | 64 bytes |
| Private Key | 32 bytes |

|  |  |
| --- | --- |
| BLOB | Size |
| Algorithm Identifier | 4 bytes |
| Key size | 4 bytes |

BLOB example:

|  |  |
| --- | --- |
| BLOB | Size |
| Algorithm Identifier | 0x45, 0x43, 0x53, 0x32 |
| Key size | 0x20, 0x00, 0x00, 0x00 (32 bytes) |

Algorithm Identifier: BCRYPT\_ECDSA\_PRIVATE\_P256\_MAGIC

For the Algorithm Identifier the above have to be used. ECDSA NISTP256 requires it.

Key size has to be 32 bytes for ECDSA NISTP256.

Public Key:

|  |  |
| --- | --- |
| Public Key | Size |
| Point X | 32 bytes Big endian |
| Point Y | 32 bytes Big endian |

Private Key:

|  |  |
| --- | --- |
| Private Key | Size |
| Point D | 32 bytes Big endian |

Description:

Before using sign or verify functions a key pair has to be imported for usage. The above structure tells you the details.

|  |  |
| --- | --- |
| Function name | dllEcdsaSign(.., .., .., ..) |
| Byte Hash[] – in | Message digest (32 bytes) |
| long HashSizeBytes | Message digest length (32) |
| Byte Signature[] – out | Output Signature (64 bytes) |
| long SignatureSizeBytes | Signature length (64) |
| Return value | CNG status from C++ |

Description:

Before using this function dllImport\_KeyPair() has to be called. The function creates a 64 bytes long signature based on ECDSA NISTP256 alogrithm.

If the return value is 0 (zero) the signature was generated with no error.

|  |  |
| --- | --- |
| Function name | dllEcdsaVerify(.., .., .., ..) |
| Byte Hash[] – in | Message digest (32 bytes) |
| long HashSizeBytes | Message digest length (32) |
| Byte Signature[] – in | Output Signature (64 bytes) |
| long SignatureSizeBytes | Signature length (64) |
| Return value | CNG status from C++ |

Description:

Before using this function dllImport\_KeyPair() has to be called. The function verifies a signature based on the input hash, signature and keypair that was previously imported with dllImport\_KeyPair() function.

If the return value is 0 (zero) the signature was verified successfully.

# Possible return values

This chapter summarizes not all, but common return error codes and possible reasons.

appSHA256\_Init()

No possible error value is expected.

appSHA256\_Update()

No possible error value is expected. If Data[] bytes array size is not consistent with noOfBytes parameter, wrong hash could be returned at appSHA256\_Finish() call.

appSHA256\_Finish()

|  |  |
| --- | --- |
| Error Code | 2989 |
| Description | The hash size is not 32 bytes. |

dllImport\_KeyPair()

|  |  |
| --- | --- |
| Error Code | 2989 |
| Description | The keypair size is not 104 bytes. |

|  |  |
| --- | --- |
| Error Code | 13 |
| Description | The keypair is not valid. Public key does not match the privatekey. |

dllEcdsaSign()

|  |  |
| --- | --- |
| Error Code | 8 |
| Description | The keypair is not valid. Public key does not match the privatekey. |

|  |  |
| --- | --- |
| Error Code | 2989 |
| Description | Hash size is not 32 bytes or signature size is not 64 bytes |

dllEcdsaVerify()

|  |  |
| --- | --- |
| Error Code | 2989 |
| Description | Hash size is not 32 bytes or signature size is not 64 bytes |

|  |  |
| --- | --- |
| Error Code | 8 |
| Description | The keypair is not valid. Public key does not match the privatekey. |

|  |  |
| --- | --- |
| Error Code | -24576 |
| Description | * + - KeyBlock[] array size was not 104 bytes.     - The keypair is not valid. Public key does not match the privatekey.     - HashSizeBytes is not 32.     - Signature verification failed (signature is not valid) |

|  |  |
| --- | --- |
| Error Code | 13 |
| Description | The keypair is not valid. Public key does not match the privatekey. |