# Using the CRYPTO Module in Traveo™ II Family

## Associated part family

Traveo™ II

## About this document

### Scope and purpose

AN220253 describes how to configure and use the cryptography block (CRYPTO) module in the Traveo™ II family. This application note covers the usage of various cryptography-related functions such as asymmetric/symmetric encryption/decryption, hash value calculation, and true and pseudo random number generation.

## Table of contents

# 1 Introduction

Embedded applications today have ever-increasing requirements regarding security features. In automotive applications, security features are typically used to do the following:

- Protect intellectual property
- Authenticate messages between electronic control units in the car or from the outside
- Provide tamper protection (e.g., prevent tuning of engine parameters or modification of trip recorder)
- Allow after-market feature activation (e.g., enable speed limiter)
- Ensure that only original equipment manufacturer (OEM) spare parts can be used

The Traveo II family of microcontrollers offers a specially developed set of security features to enable such use cases. Apart from life cycle and protection schemes such as those that control the access permissions for the debug interface or for the software running on the microcontroller, one important component is the cryptography block (or CRYPTO) which will be explained in this application note. These descriptions will focus on the Infineon “Crypto” driver, part of the Sample Driver Library (SDL).

# 2 Cryptographic Operations and Features

## 2.1 Random Number Generator (RNG)

There are two types: pseudo random number generator (PRNG) and true random number generator (TRNG).

A PRNG outputs a random looking sequence of numbers based on a mathematical formula and a given start value (also known as “seed”). Attackers with knowledge about the PRNG may be able to predict future random numbers.

TRNGs use physical properties (thermal noise, for example) as a source of true randomness. Typically, post-processing steps are executed and monitoring features are implemented to detect a malfunctioning of the TRNG, which might be caused by an attacker trying to override the output of a TRNG.

The advantages of PRNGs are that they are usually much faster in providing random numbers than the TRNG. In addition – depending on the physical implementation – active TRNGs may cause power dissipation that is unacceptable by certain applications (for example, a ring oscillator causes dynamic switching currents). These are the reasons why many applications use a TRNG to generate a true random seed for the PRNG.

## 2.2 Symmetric Cryptography

Symmetric cryptography relies on a shared secret between the sender and receiver of a message. The same secret key is used for encryption as well as decryption.

## 2.3 Asymmetric Cryptography

In the case of asymmetric cryptography, each party has a private and public key. The principles of asymmetric cryptography allow a sender to use the receiver’s public key to encrypt a message that can be decrypted by the receiver with its private key. Alternatively, the sender can generate a digital signature of a message using its private key and any receiver can verify this message by using the sender’s public key.

## 2.4 Hash Functions

Hash functions produce a fixed-length output (“hash value” or “digest”) for a variable length input. Cryptographic hash functions are a special kind of hash functions with certain stronger requirements related to their characteristics.

The hash value together with the input data to the hash functions can be used to check the integrity of the input data or hash value.

## 2.5 Message Authentication Codes (MAC)

MACs are based on cryptographic hash functions or symmetric cryptography. In addition to the integrity of a message, the authenticity of a message can be ensured because of a shared secret between sender and receiver.

## 2.6 Digital Signatures

Digital signatures are generated by using asymmetric cryptography in combination with cryptographic hash functions. They provide the following security features: integrity, authenticity, and non-repudiation, i.e., the sender cannot deny that a message originated from them because the sender’s private key was used to generate the signature and the receiver can prove this using the sender’s public key.

# 3 Crypto Driver

## 3.1 Driver Architecture

The design of the Crypto driver is based on a client-server architecture. The Crypto server runs only on the CM0+ core and works with the Crypto hardware by executing appropriate Crypto Core functions. The Crypto client can be run on either core, but this application note only covers the case that the client runs on the CM4 core. All descriptions and figures that refer to CM4 are also applicable for Traveo II devices with (dual) CM7. The client and server communicate through the Inter Processor Communication (IPC). Using IPC for communication provides a simple synchronization mechanism to handle concurrent requests from different cores.

The Crypto driver utilizes the following hardware resources besides the CRYPTO block:

- 1 IPC channel structure for data exchange between client and server
- 2 IPC interrupt structures for notifications
- 1 interrupt for handling of hardware errors

The general communication concept between Crypto server and client is depicted in **Figure 1**.

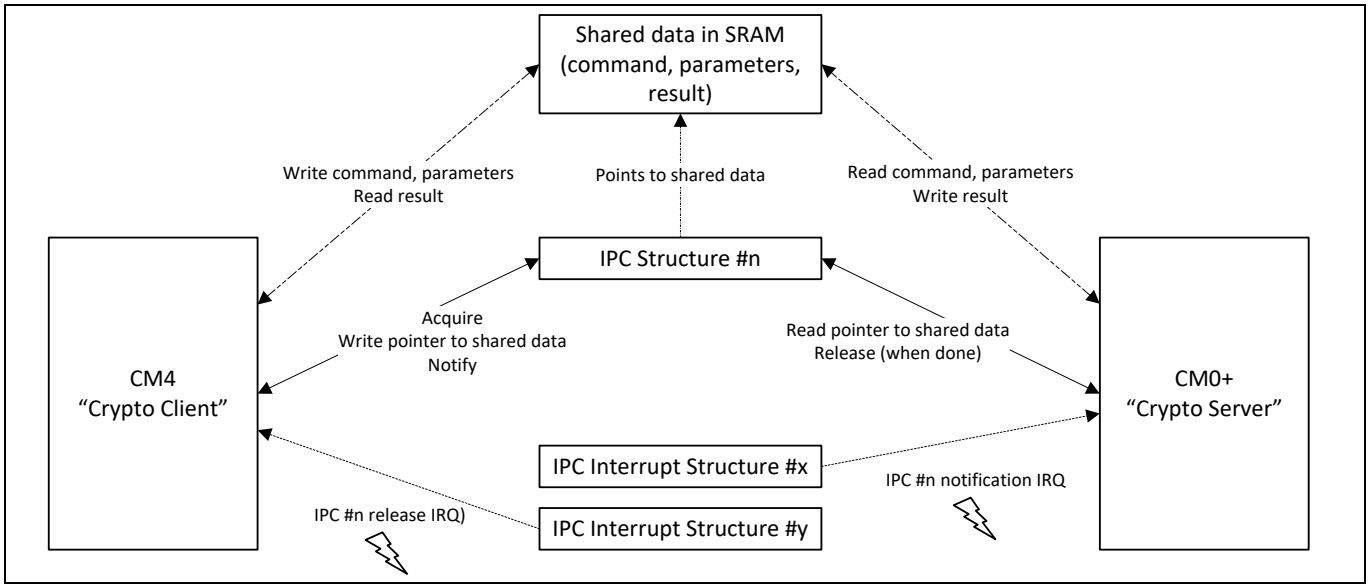


**Figure 1 Server/Client Communication Example via IPC**

## 3.2 Driver Initialization

Before the driver can be used for cryptographic operations, it needs to be initialized in the correct sequence.

1. The Crypto server must be initialized by CM0+.
2. After the Crypto server initialization is completed, the client can be successfully initialized by CM4.

*Note: Because the Crypto server runs on CM0+ and the client can run on CM4 and both cores run independently from each other with different frequencies, server initialization may not be finished when client is initialized. Ideally, the CM4 core is released from reset after the Crypto server has been initialized by CM0+ to avoid this situation. Alternatively, the server state can be checked before initializing the client.*

**Figure 2** shows the initialization of the Crypto server and client.

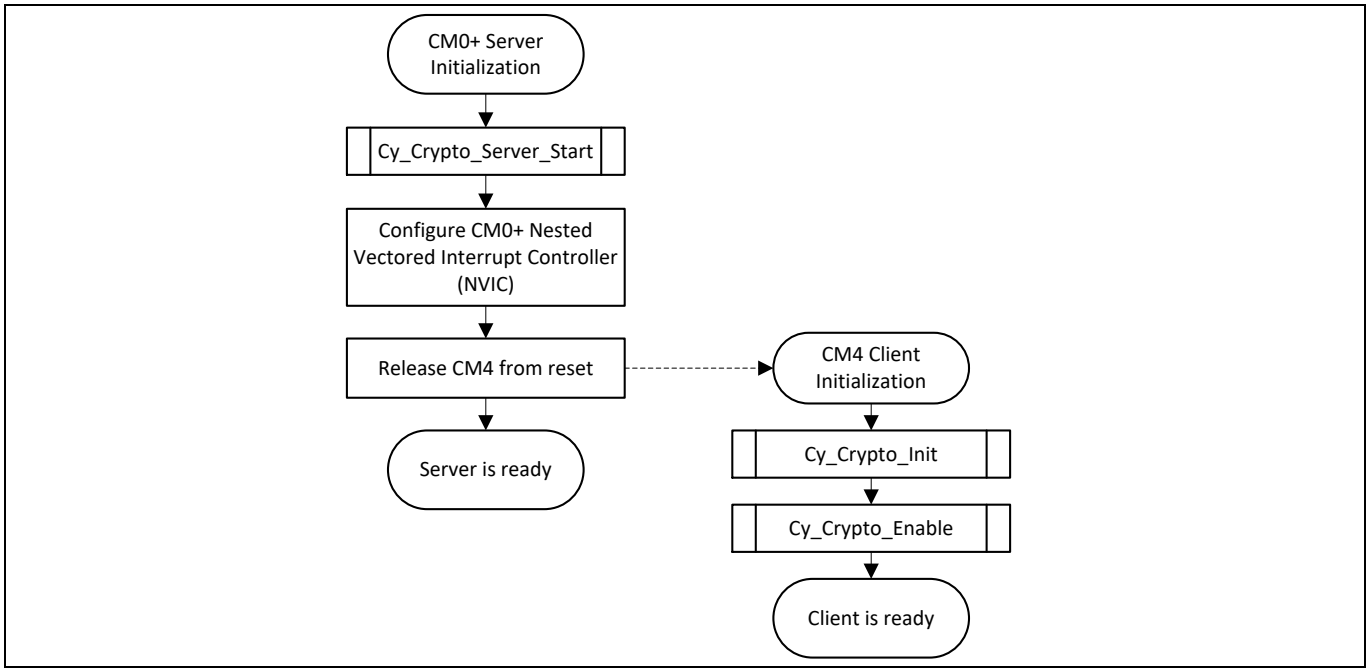


**Figure 2 Server/Client Initialization Flowchart Example**

## 3.3 Driver Usage

Once the initialization of the server as well as the client is complete, the client can request cryptographic operations from the server. While the server running on CM0+ is processing the requested operation, CM4 can be used to process other tasks. The end of the operation can be determined by three different methods:

1. Calling the crypto driver function `Cy_Crypto_Sync` in blocking mode. Once the function returns, the client can be sure that the previous operation is complete.
2. Periodically calling the crypto driver function `Cy_Crypto_Sync` in non-blocking mode and checking the return value. If the return value indicates that the server is ready, the previous operation is complete
3. The client can configure the IPC release interrupt to get notified immediately upon completion of the server operation.

In this application note, the first method (calling `Cy_Crypto_Sync` in blocking mode) is used and shown in the flowcharts, also indicated by the function parameter and value “block=true”. **Figure 3** shows the general usage of the Crypto driver once it has been initialized.

## Crypto Driver

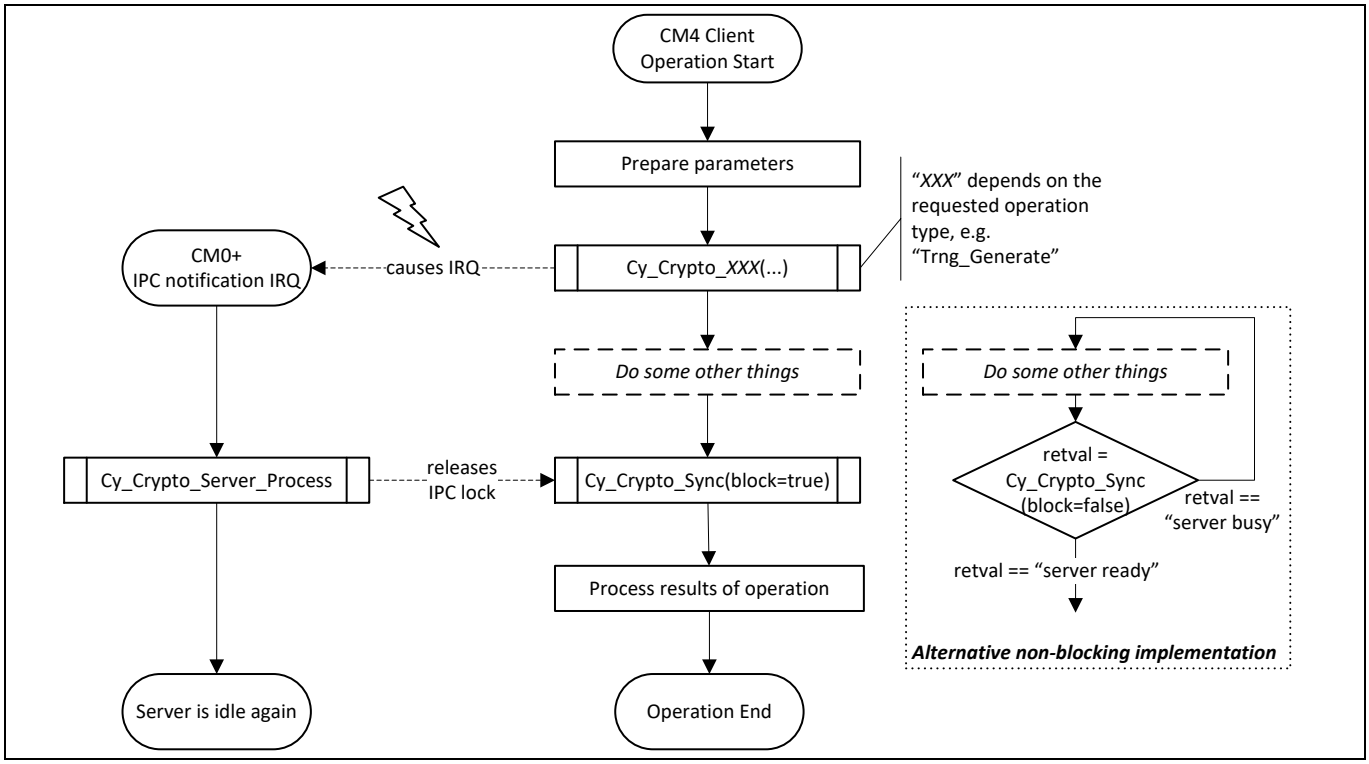


**Figure 3 General Driver Usage Flowchart Example**

# 4 Cyclic Redundancy Check (CRC)

## 4.1 Use Case

The CRC result of a memory area (for example, parameters in flash) can be stored in addition to this area. Software can then ensure that the contents of the area have not been corrupted by causes such as environmental influences or hardware defects.

*Note:* *CRC is not a cryptographic operation and must not be used to protect such areas against malicious modifications, because an attacker can easily compute the CRC result for modified areas.*

## 4.2 Driver Functions

The Crypto driver provides following CRC-related functions:

### 4.2.1 Cy_Crypto_Crc_Init

This function initializes basic CRC-related settings that can later be reused across multiple CRC calculations. Many CRC standards such as the widely used CRC32 or CRC16-CCITT differ in the used polynomial (and length) and input/output settings. Those settings are defined by calling `Cy_Crypto_Crc_Init`.

### 4.2.2 Cy_Crypto_Crc_Run

The actual CRC calculation is requested by calling this function. The initial value for calculation, start address, and size are passed to the function and the CRC result is returned.

The initial value for the CRC calculation is defined by the used CRC standard and applies for the start of a new CRC calculation. If CRC is calculated over multiple not-contiguous memory areas, `Cy_Crypto_Crc_Run` must be called multiple times (after the current operation has finished) and the result of the previous block must be used as the initial value for the next block.

## 4.3 Flowchart

**Figure 4** shows the general flow of how to use the Crypto driver to calculate a CRC value.

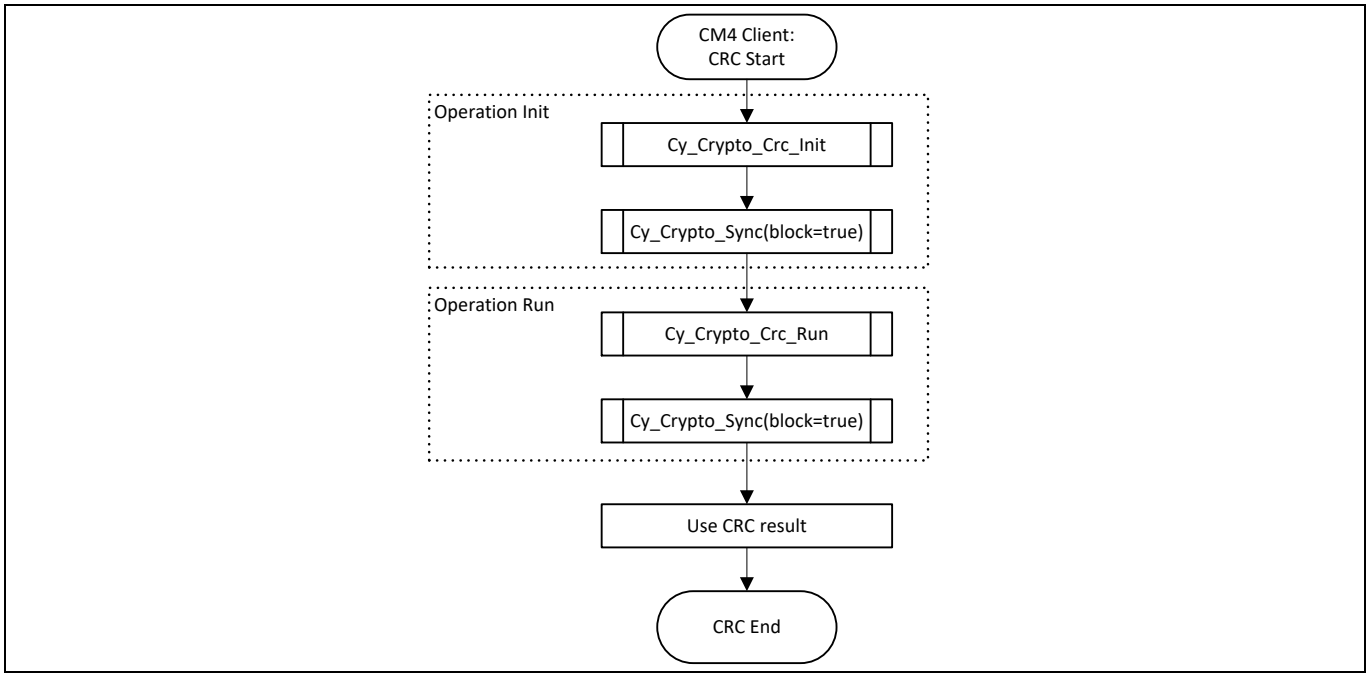


**Figure 4 Flowchart of CRC Calculation**

*Note: For repeated usage of CRC operations, it is not necessary to call `Cy_Crypto_Crc_Init` again unless one of the related parameters (e.g. polynomial) changes.*

# 5 Pseudo Random Number Generator (PRNG)

## 5.1 Use Case

The pseudo random number generator can be used to derive the following:

- symmetric keys for a communication session
- "cryptographic salt", an additional random input for hash functions to prevent dictionary attacks that use huge databases of pre-computed hash values for all words in a dictionary
- “challenge” values in challenge-response-authentication protocols.

*Note:* *The PRNG seed values should be initialized with true random values with high entropy to provide strong security characteristics. TRNG can be used for that purpose. Non-random or constant seed values should only be used if a deterministic behavior such as reproducible PRNG output is required in temporary situations such as during software development and testing.*

## 5.2 Driver Functions

The Crypto driver provides the following PRNG related functions:

### 5.2.1 Cy_Crypto_Prng_Init

This function takes the seed values to initialize the three linear feedback shift registers that constitute the PRNG.

### 5.2.2 Cy_Crypto_Prng_Generate

A pseudo random number is generated by calling this function. The upper limit for the pseudo random number can be specified so that the function returns a number between 0 and the specified limit. The maximum upper limit is $2^{32}$ - 1.

*Note:* `Cy_Crypto_Prng_Init` *needs to be called only when the seed values is changed afterwards, it is sufficient to call* `Cy_Crypto_Prng_Generate` *when multiple random numbers are needed.*

## 5.3 Flowchart

**Figure 5** shows the general flow of how to use the Crypto driver to generate a pseudo random number.

## Pseudo Random Number Generator (PRNG)

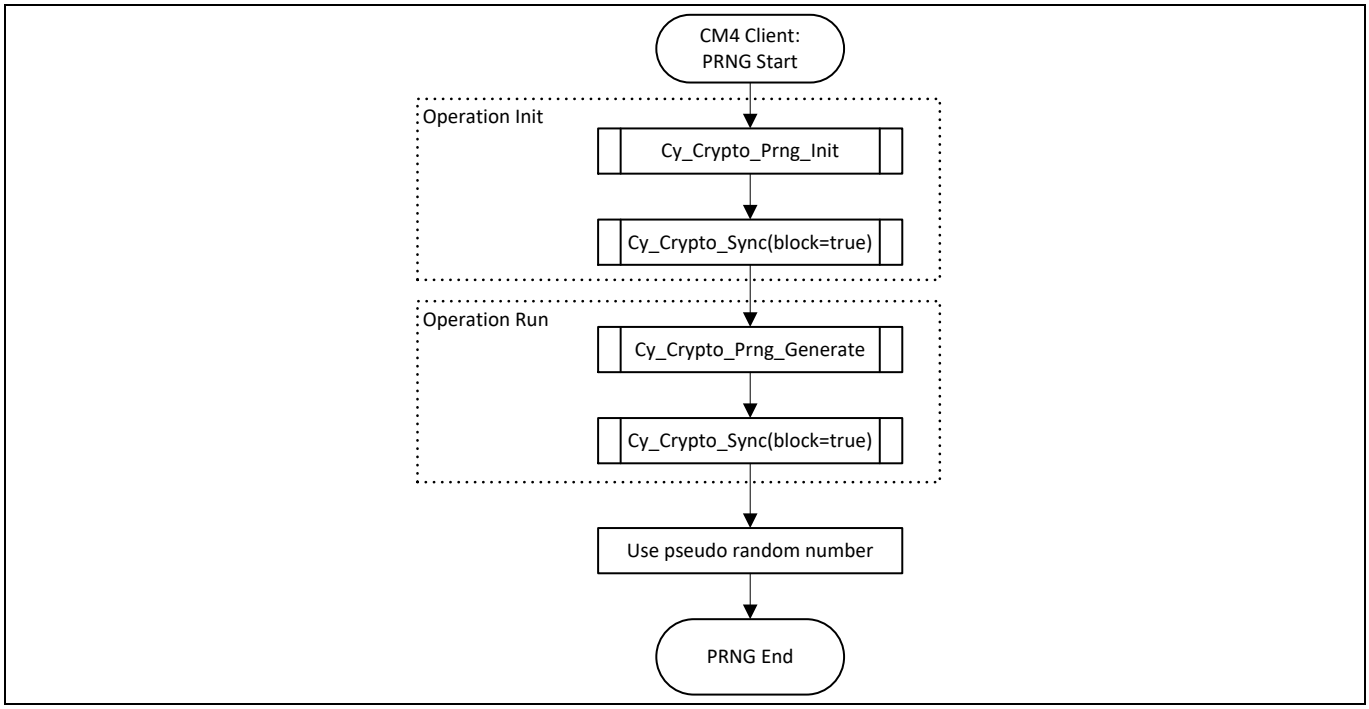


**Figure 5 Flowchart of Pseudo Random Number Generation**

*Note:* `Cy_Crypto_Prng_Init` *needs to be called only if the seed values is changed; this function need not be called for repeated generation of PRNG values.*

# 6 True Random Number Generator (TRNG)

## 6.1 Use Case

TRNG is typically used only to generate true random seed values with high entropy for PRNG because it has a comparably high current consumption and produces the random bits rather slowly.

## 6.2 Driver Functions

The Crypto driver provides the following TRNG related functions:

### 6.2.1 Cy_Crypto_Trng_Generate

This function generates a true random number with the specified bit length. Some parameters related to its physical construction (ring oscillator) need to be stated when calling this function.

## 6.3 Flowchart

**Figure 6** shows the general flow of how to use the Crypto driver to generate a true random number.

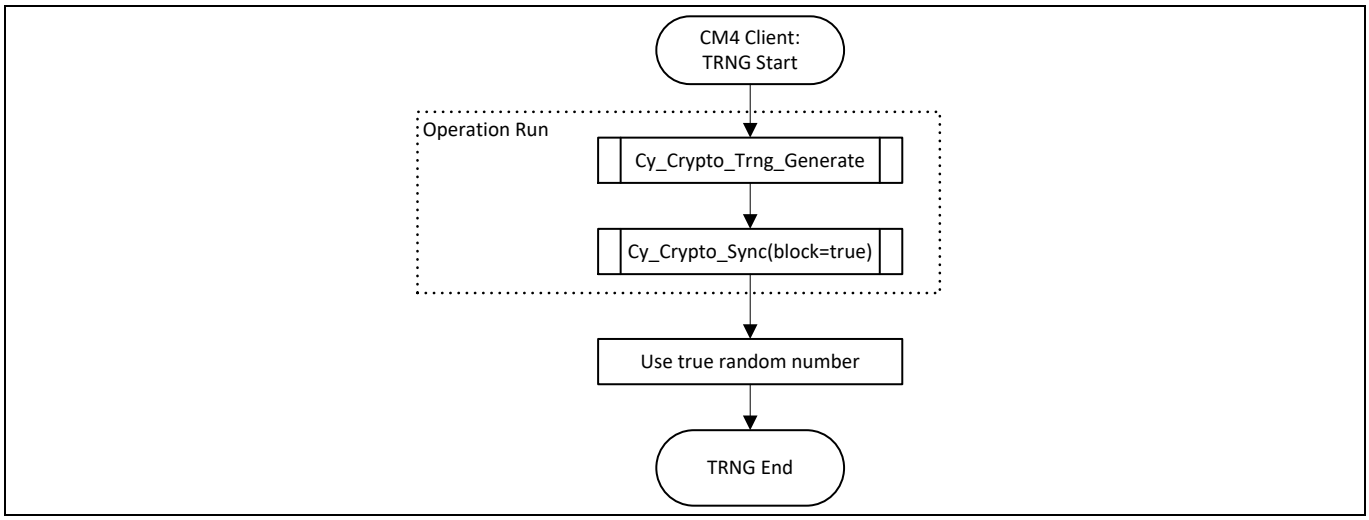


**Figure 6 Flowchart of True Random Number Generation**

# 7 Symmetric Key Cryptography Using Advanced Encryption Standard (AES)

The Crypto block supports several symmetric key cryptography standards, but AES is the most popular and widely used one.

AES operations per se are based on a single block of data of 128 bits. If multiple blocks of related data are encrypted, it is recommended to use one of the block chaining modes that have been described in scientific publications such as **National Institute of Standards and Technology (NIST) Special Publication 800-38A**. Encrypting each block independently usually provides an insufficient level of security-- an example is the case where blocks with the same plaintext would also have the same cipher text.

The Crypto driver supports the following AES operation modes:

- ECB (Electronic Code Book Mode)
  This mode encrypts/decrypts blocks independently without chaining and therefore provides a low level of security for multiple blocks of related data.
- CBC (Cipher Block Chaining Mode)
  The cipher text of one block is combined with the plaintext of the following block. Because the first block does not have a preceding block, a so-called initialization vector (IV) is needed, which is then combined with the plaintext of the first block.
- CFB (Cipher Feedback Mode)
  This mode can be used as stream cipher for a plaintext with arbitrary length. The plaintext is XORed with the output of the AES encryption operation. The cipher text is used as the encryption input for the succeeding block. The first block requires an initialization vector similar to CBC mode.
- CTR (Counter Mode)
  The counter mode of operation is also used as a stream cipher. Again, the plaintext is XORed with the output of the AES encryption operation. The input to the AES encryption is a combination or concatenation of a so-called nonce and a counter. The counter changes for every block; in the simplest case, it is just being incremented.
  A nonce in the cryptographic context is an arbitrary number that is usually only used once for a cryptographic operation.

## 7.1 Use Case

Because the performance requirements of AES are so much lower than for asymmetric key cryptography, it is typically used for secure communication between devices that possesses the cryptographic key. In contrast, asymmetric key cryptography is often only used by communication protocols to establish the communication session by exchanging the key between all involved parties. This key will then be used for symmetric encryption and decryption.

## 7.2 Driver Functions

The Crypto driver provides following AES related functions. An initialization of AES is not needed.

### 7.2.1 Cy_Crypto_Aes_Ecb_Run

This function encrypts or decrypts a single 128-bit block of data. The user is requested to specify the direction (encryption/decryption), key location and key size, and source and destination block location.

### 7.2.2 Cy_Crypto_Aes_Cbc_Run

In addition to `Cy_Crypto_Aes_Ecb_Run`, this function expects the initialization vector and the total size of the data to be encrypted/decrypted.

### 7.2.3 Cy_Crypto_Aes_Cfb_Run

This function expects the same arguments as `Cy_Crypto_Aes_Cbc_Run`.

### 7.2.4 Cy_Crypto_Aes_Ctr_Run

This function expects the same arguments as `Cy_Crypto_Aes_Cbc_Run`. The initialization vector is the nonce combined with the starting counter value. The driver internally increments the counter part for every new block encryption.

## 7.3 Flowchart

**Figure 7** shows the general flow of how to use the Crypto driver to execute AES operations.

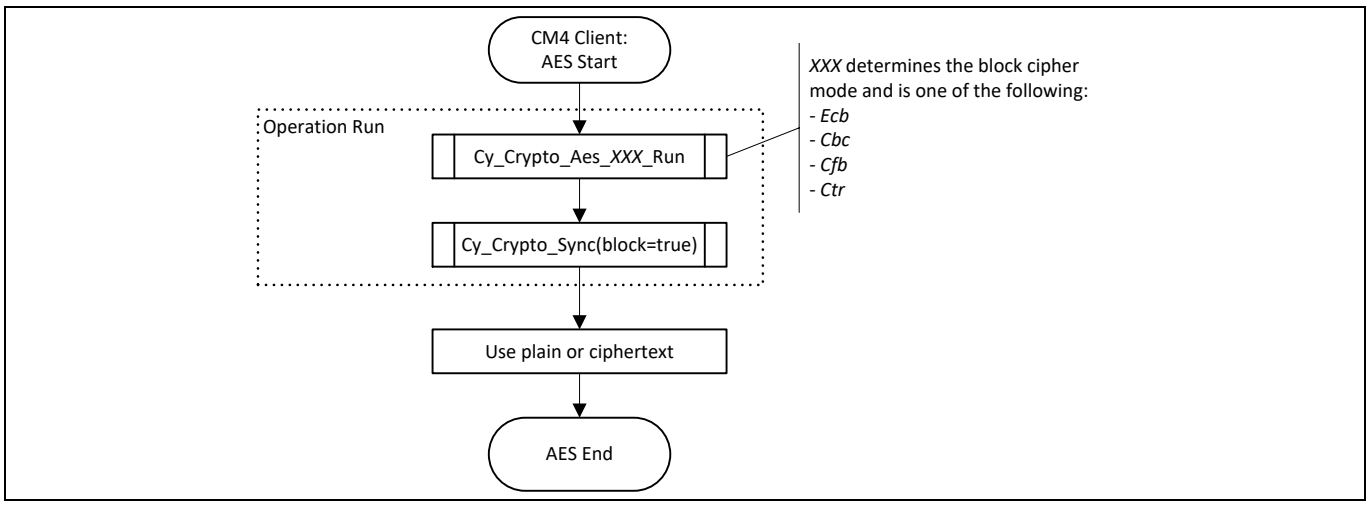


**Figure 7 Flowchart of AES Operation**

## 7.4 Other Symmetric Key Algorithms

In addition to the popular AES algorithm, the Crypto driver and hardware support the following algorithms as well:

- DES (Data Encryption Standard)
  The `Cy_Crypto_Des_Run` function is offered by the Crypto driver.
- TDES (Triple DES, sometimes also abbreviated as 3DES)
  The corresponding driver function is `Cy_Crypto_Tdes_Run`.
- ChaCha
  ChaCha is supported by the `Cy_Crypto_Chacha_Run` driver function.

# 8 SHA Family of Cryptographic Hash Functions

SHA stands for Secure Hash Algorithm and is a standardized family of cryptographic hash functions. The Crypto driver supports all standardized variants such as SHA-1, SHA-256, SHA-512, SHA-512/256, SHA3-256, SHA3-512, SHAKE256, and many more.

## 8.1 Use Case

SHA is often used to derive a hash value from a user password (ideally combined with a "cryptographic salt value") and store this hash value (together with the salt) in a database instead of directly storing the actual plaintext password. In the event of a security breach, the database will not reveal the user password which could be used to log in to other systems.

SHA is also used in combination with other cryptographic building blocks to achieve higher level security goals: for example, in hash-based message authentication codes (see **Hash-based and Cipher-based Message Authentication Code (HMAC, CMAC)**) or digital signatures (see **Digital Signature Verification Using RSA and SHA**).

## 8.2 Driver Functions

The Crypto driver provides the following SHA related functions. An initialization of SHA is not needed.

### 8.2.1 Cy_Crypto_Sha_Run

This function generates the message digest per the chosen SHA mode, which needs to be specified when calling the function. In addition, the location of the message, its size, and the location where to store the digest are expected by the function.

## 8.3 Flowchart

**Figure 8** shows the general flow of how to use the Crypto driver to execute SHA operations.

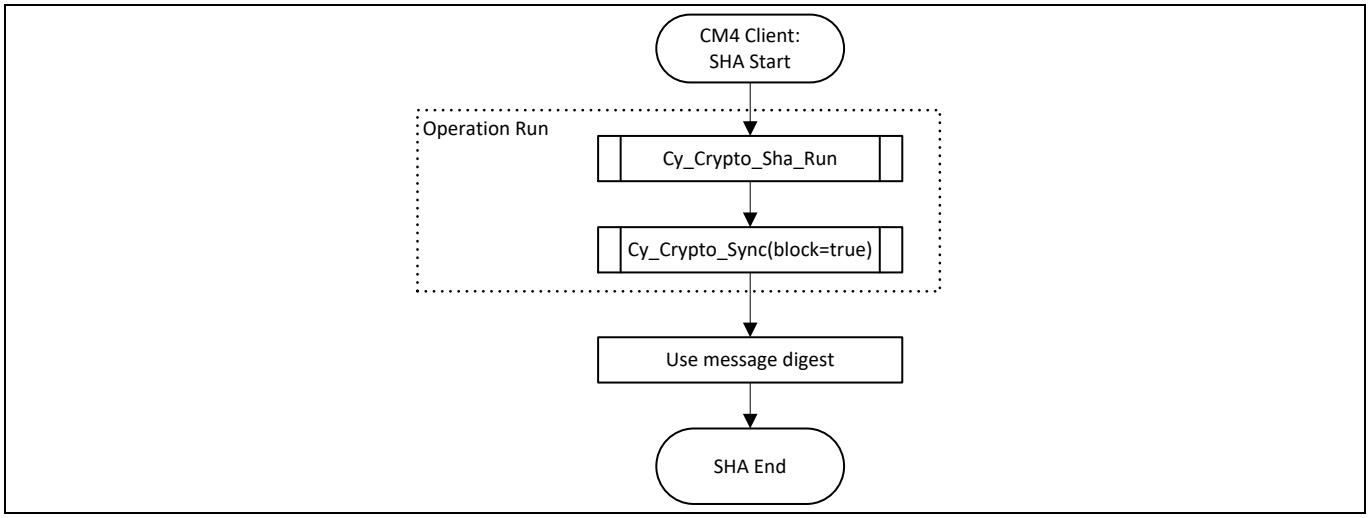


**Figure 8 Flowchart of SHA Operation**

# 9 Hash-based and Cipher-based Message Authentication Code (HMAC, CMAC)

Message Authentication Codes (MAC) are similar to cryptographic hash functions but have different security requirements. These security requirements can be achieved by using a secret cryptographic key together with either a cryptographic hash function or a block cipher as the basis for generating the MAC. The operation is then called Hash-based MAC (HMAC), or Cipher-based MAC (CMAC), respectively. The Crypto driver uses the SHA block of the Crypto hardware for HMAC operations and the AES block for CMAC operations.

## 9.1 Use Case

MACs are used whenever it is important for the receiver of a message to know that the message originated from the authentic sender. An example is where distributed electronic control units in a car network can ensure that no attacker can inject messages to the network by using MACs for all critical messages. This is achieved by a shared secret key between the sender and receiver, which is used in the MAC generation and verification process.

## 9.2 Driver Functions

The Crypto driver provides the following MAC-related functions. An initialization of MAC operations is not needed.

### 9.2.1 Cy_Crypto_Hmac_Run

When calling this function, you should pass the desired SHA mode, key location and size, message location and size, and location where the calculated MAC will be stored.

### 9.2.2 Cy_Crypto_Cmac_Run

This function expects similar parameters as `Cy_Crypto_Hmac_Run` but uses the SHA mode.

## 9.3 Flowchart

**Figure 9** shows the general flow of how to use the Crypto driver to execute MAC operations.

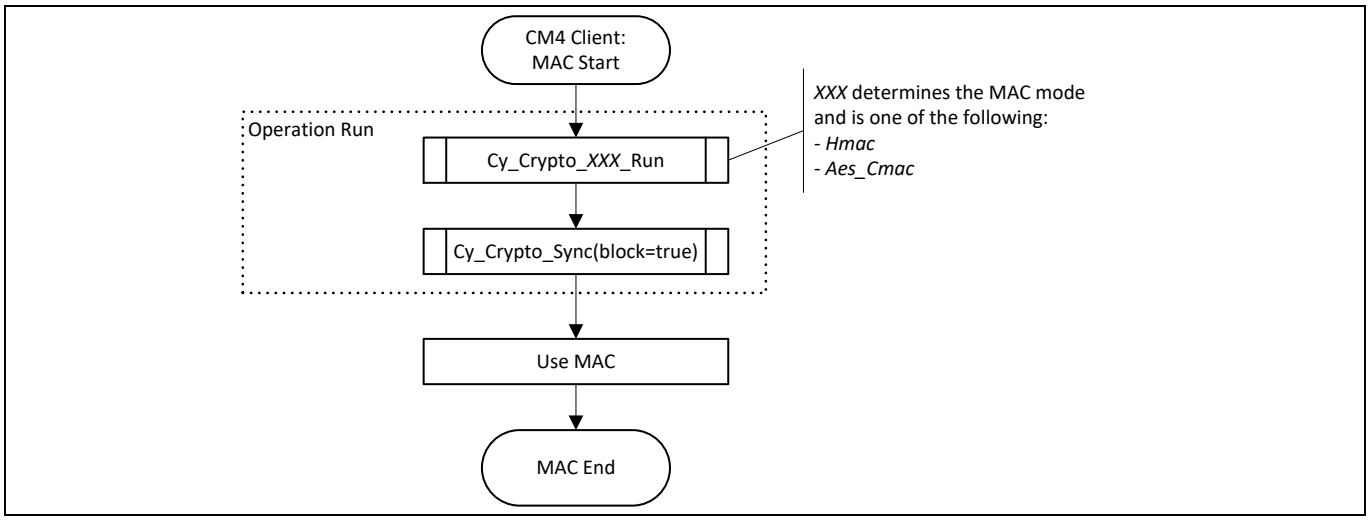


**Figure 9 Flowchart of MAC Operations**

# 10 Asymmetric Key Cryptography Using RSA

Various algorithms for asymmetric key cryptography (also known as public key cryptography) have been proposed in the past decades and are used by the public. One of the most popular ones is RSA, named by its inventors Rivest, Shamir, and Adleman. The Crypto driver supports RSA algorithm.

Verifying digital signatures based on RSA and SHA requires combined usage of Crypto driver APIs related to RSA and SHA. See **Digital Signature Verification Using RSA and SHA**.

## 10.1 Use Case

Encryption and decryption using asymmetric cryptography is typically a lot slower than symmetric encryption or decryption. This is the reason why it is often only used during establishment of a communication channel to exchange a symmetric key between all involved parties.

## 10.2 Driver functions

The Crypto driver provides the following functions related to RSA encryption and decryption.

### 10.2.1 Cy_Crypto_Rsa_InvertEndianness

The multi-byte parameters needed for RSA operations such as plaintext, cipher text, modulus, and public and private exponent are usually defined in big endian format. The Crypto driver internally works with little endian format, so one needs to ensure that all input parameters are available in little endian format. This conversion can be done in software during runtime by calling `Cy_Crypto_Rsa_InvertEndianness`. The Crypto driver stores the output (plaintext or cipher text) of `Cy_Crypto_Rsa_Proc` in little endian format; you can use this function to,convert the output to big endian format if necessary.

### 10.2.2 Cy_Crypto_Rsa_CalcCoefs

This function can pre-calculate certain coefficients and parameters used by the RSA calculations that will speed up RSA processing. Calculated coefficients are attached to the RSA-related Crypto driver context and can be reused across multiple RSA operations as long as dependent parameters such as modulus do not change.

If the function is not called during RSA operation pre-processing, `Cy_Crypto_Rsa_Proc` will internally calculate these parameters every time it is called.

### 10.2.3 Cy_Crypto_Rsa_Proc

This is the main function for encrypting plaintexts or decrypting cipher texts using RSA.

The function interface is generic; it takes an input message location and its size, an output message location, the modulus, a key (i.e., exponent), and optionally the location of the coefficients calculated by `Cy_Crypto_Rsa_CalcCoefs`.

The `Cy_Crypto_Rsa_Proc` function processes the provided input message with the modulo and exponent and stores the output at the output message location. It depends on the type of the input message provided (plaintext or cipher text) and the type of exponent provided (public or private). The function internally does not distinguish between the operations, whether encryption or decryption.

## 10.3 Flowchart

**Figure 10** shows the general flow of how to use the Crypto driver for RSA encryption and decryption operations.

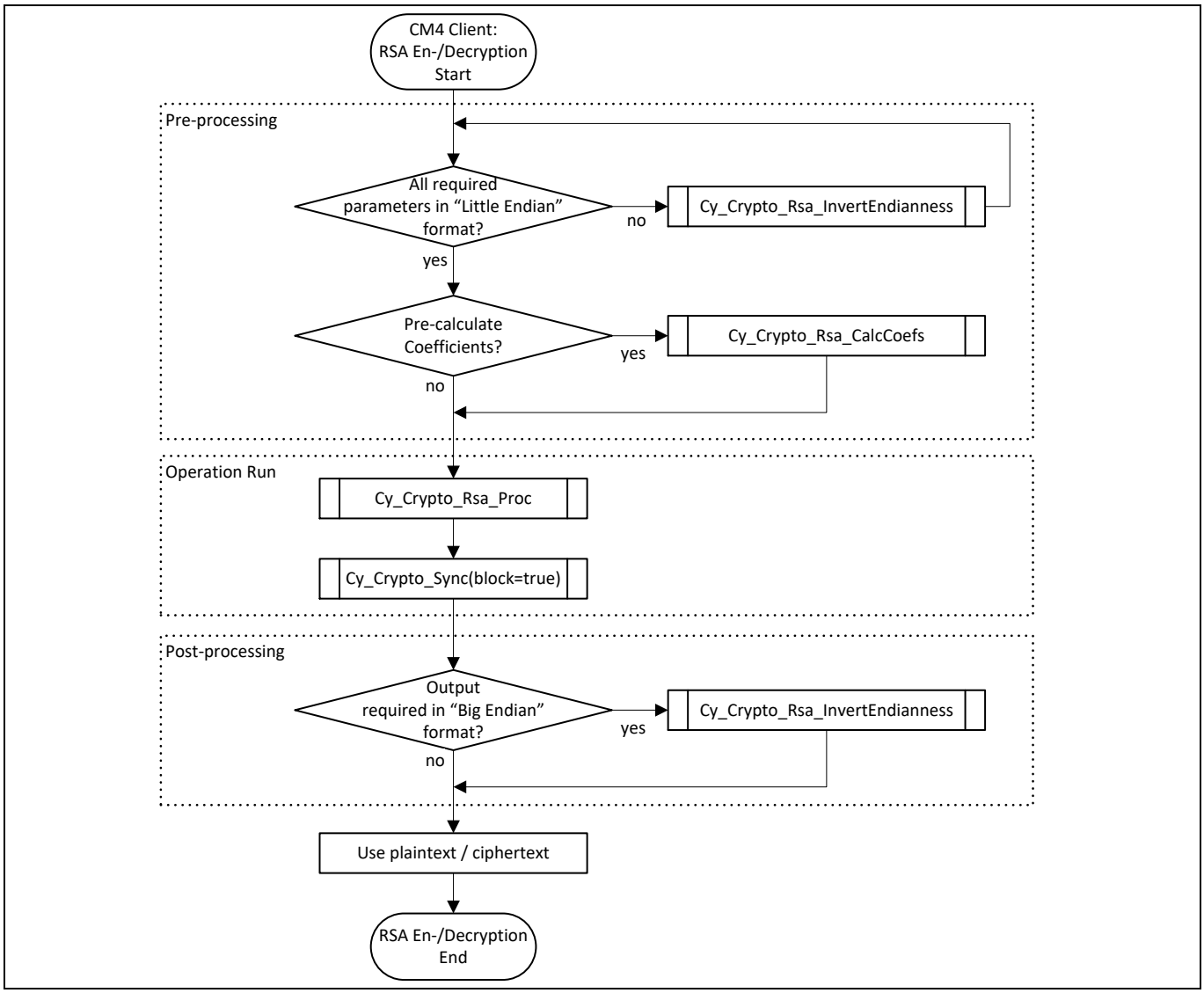


**Figure 10 Flowchart of RSA Encryption and Decryption Operations**

# 11 Digital Signature Verification Using RSA and SHA

Digital signatures are widely used and often based on RSA and a hash function with cryptographic strength.

The Public-Key Cryptography Standards (PKCS) is a collection of specifications that tries to standardize various aspects of asymmetric key cryptography. One of those aspects is to define how digital signatures shall be generated and verified. As of now, the Crypto driver supports only verification of digital signatures based on RSASSA-PKCS1-v1_5 and only using SHA cryptographic hash functions. More information can be found in following document: **http://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf**.

## 11.1 Use Case

One of the application fields of digital signatures is the verification of firmware images before either installing or executing them. For example, the firmware update process will receive a firmware image and related digital signature over an insecure communication channel. The update process will then be able to prove the authenticity and origin of the firmware image by verifying the digital signature using the public key that was programmed to the device by the manufacturer during production.

## 11.2 Driver Functions

The Crypto driver provides the following functions related to verifying digital signatures using RSA and SHA.

### 11.2.1 Cy_Crypto_Rsa_InvertEndianness

See **Cy_Crypto_Rsa_InvertEndianness**.

### 11.2.2 Cy_Crypto_Rsa_CalcCoefs

See **Cy_Crypto_Rsa_CalcCoefs**.

### 11.2.3 Cy_Crypto_Rsa_Proc

See **Cy_Crypto_Rsa_Proc**.

Verification of a digital signature requires that the signature is decrypted using this function and the public key of the signer.

### 11.2.4 Cy_Crypto_Sha_Run

See **Cy_Crypto_Sha_Run**.

The same SHA mode, that was used in the generation of the digital signature, must be used to calculate the digest of the message, that has been signed.

### 11.2.5 Cy_Crypto_Rsa_Verify

This function does the actual verification. It takes the decrypted digital signature, validates the format of that signature according to the supported RSASSA-PKCS1-v1_5 standard and checks whether the provided message digest matches the one included in the digital signature.

## 11.3 Flowchart

**Figure 11** shows the general flow of how to use the Crypto driver, in order to verify digital signatures using RSA and SHA operations.

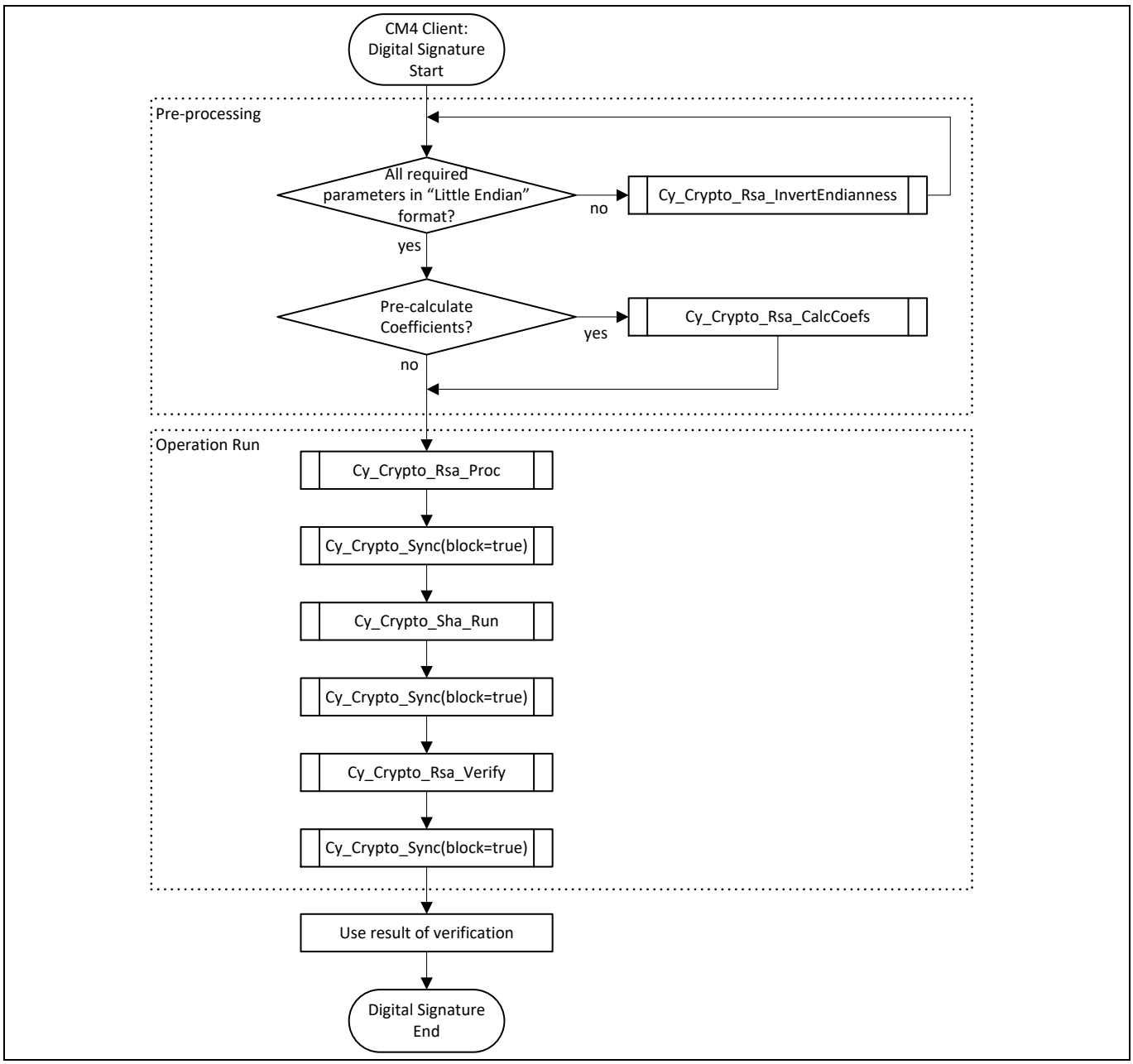


**Figure 11 Flowchart of Verifying Digital Signatures with RSA and SHA Operations**

# Revision history

| Document version | Date of release | Description of changes |
|---|---|---|
| ** | 01/25/2018 | New application note |
| *A | 2021-02-15 | Moved to Infineon Template |

**Trademarks**
All referenced product or service names and trademarks are the property of their respective owners.

**Edition 2021-02-15**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**



**Do you have a question about this document?**

**Go to: www.cypress.com/support**

**Document reference**

**002-20253 Rev. *A**

**IMPORTANT NOTICE**

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

**WARNINGS**

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.