

Xilinx Standalone Library Documentation

XilSecure Library v4.2

UG1189 (v2020.1) June 3, 2020



Table of Contents

Chapter 1: Overview.....	3
Chapter 2: AES-GCM.....	4
XilSecure Zynq UltraScale+ MPSoC AES APIs.....	4
AES-GCM Error Codes.....	12
AES-GCM API Example Usage.....	13
AES-GCM Usage to decrypt Boot Image.....	14
Chapter 3: RSA.....	16
Functions.....	17
Data Structures.....	20
RSA API Example Usage.....	21
Chapter 4: SHA-3.....	23
Functions.....	24
SHA-3 API Example Usage.....	29
Chapter 5: XilSecure Utilities.....	31
Functions.....	32
Appendix A: Additional Resources and Legal Notices.....	35
Xilinx Resources.....	35
Documentation Navigator and Design Hubs.....	35
Please Read: Important Legal Notices.....	36

Overview

The XilSecure library provides APIs to access cryptographic accelerators on the Zynq UltraScale+ MPSoC devices. The library is designed to run on top of Xilinx standalone BSPs. The library is tested on the following Zynq UltraScale+ MPSoC devices:

- A53
- R5
- MicroBlaze

XilSecure is used during the secure boot process. The XilSecure library includes:

- SHA-3/384 engine for 384 bit hash calculation.
- AES-GCM engine for symmetric key encryption and decryption using a 128-bit and 256-bit key.
- RSA engine for signature generation, signature verification, encryption and decryption. Key sizes supported include 2048, 3072, and 4096.



CAUTION! *The Vitis unified software platform defaults to using a software stack in DDR and any variables used by XilSecure will be placed in DDR. For better security, change the linker settings to make sure the stack used by XilSecure is either in the OCM or the TCM.*

Note: The XilSecure library does not check for memory bounds while performing cryptographic operations. You must check the bounds before using the functions provided in this library. If needed, you can take advantage of the XMPU, the XPPU, and/or TrustZone to limit memory access.

Source Files

The source files for the library can be found at: https://github.com/Xilinx/embeddedsw/tree/master/lib/sw_services/xilsecure/src/versal

AES-GCM

This software uses AES-GCM hardened cryptographic accelerator to encrypt or decrypt the provided data and requires a key of size 256 bits and initialization vector(IV) of size 96 bits.

XilSecure Zynq UltraScale+ MPSoC AES APIs

This software uses AES-GCM hardened cryptographic accelerator to encrypt or decrypt the provided data and requires a key of size 256 bits and initialization vector(IV) of size 96 bits.

XilSecure library supports the following features:

- Encryption of data with provided key and IV
- Decryption of data with provided key and IV
- Authentication using a GCM tag.
- Key loading based on key selection, the key can be either the user provided key loaded into the KUP key or the device key used during boot.

For either encryption or decryption the AES-GCM engine should be initialized first using the `XSecure_AesInit` function.

AES Encryption Function Usage

When all the data to be encrypted is available, the `XSecure_AesEncryptData()` can be used. When all the data is not available, use the following functions in the suggested order:

1. `XSecure_AesEncryptInit()`
2. `XSecure_AesEncryptUpdate()` - This function can be called multiple times till input data is completed.

AES Decryption Function Usage

When all the data to be decrypted is available, the `XSecure_AesDecryptData()` can be used. When all the data is not available, use the following functions in the suggested order:

1. `XSecure_AesDecryptInit()`

2. `XSecure_AesDecryptUpdate()` - This function can be called multiple times till input data is completed.

During decryption, the passed in GCM tag will be compared to the GCM tag calculated by the engine. The two tags are then compared in the software and returned to the user as to whether or not the tags matched.



CAUTION! when using the KUP key for encryption/decryption of the data, where the key is stored should be carefully considered. Key should be placed in an internal memory region that has access controls. Not doing so may result in security vulnerability.

Table 1: Quick Function Reference

Type	Name	Arguments
s32	<code>XSecure_AesInitialize</code>	XSecure_Aes * InstancePtr XCsuDma * CsuDmaPtr u32 KeySel Iv Key
u32	<code>XSecure_AesDecryptInit</code>	XSecure_Aes * InstancePtr u8 * DecData u32 Size u8 * GcmTagAddr
s32	<code>XSecure_AesDecryptUpdate</code>	XSecure_Aes * InstancePtr u8 * EncData u32 Size
s32	<code>XSecure_AesDecryptData</code>	XSecure_Aes * InstancePtr u8 * DecData u8 * EncData u32 Size
s32	<code>XSecure_AesDecrypt</code>	XSecure_Aes * InstancePtr const u8 * Src u8 * Dst u32 Length
u32	<code>XSecure_AesEncryptInit</code>	XSecure_Aes * InstancePtr u8 * EncData u32 Size
u32	<code>XSecure_AesEncryptUpdate</code>	XSecure_Aes * InstancePtr const u8 * Data u32 Size

Table 1: Quick Function Reference (cont'd)

Type	Name	Arguments
u32	XSecure_AesEncryptData	XSecure_Aes * InstancePtr u8 * Dst const u8 * Src u32 Len
void	XSecure_AesReset	XSecure_Aes * InstancePtr

Functions

XSecure_AesInitialize

This function initializes the instance pointer.

Note: All the inputs are accepted in little endian format but the AES engine accepts the data in big endian format, The decryption and encryption functions in xsecure_aes handle the little endian to big endian conversion using few API's, Xil_Htonl (provided by Xilinx xil_io library) and XSecure_AesCsuDmaConfigureEndiannes for handling data endianness conversions. If higher performance is needed, users can strictly use data in big endian format and modify the xsecure_aes functions to remove the use of the Xil_Htonl and XSecure_AesCsuDmaConfigureEndiannes functions as required.

Prototype

```
s32 XSecure_AesInitialize(XSecure_Aes *InstancePtr, XCsuDma *CsuDmaPtr, u32
KeySel, u32 *IvPtr, u32 *KeyPtr);
```

Parameters

The following table lists the XSecure_AesInitialize function arguments.

Table 2: XSecure_AesInitialize Arguments

Type	Name	Description
XSecure_Aes *	InstancePtr	Pointer to the XSecure_Aes instance.
XCsuDma *	CsuDmaPtr	Pointer to the XCsuDma instance.
u32	KeySel	Key source for decryption, can be KUP/device key <ul style="list-style-type: none"> XSECURE_CSU_AES_KEY_SRC_KUP :For KUP key XSECURE_CSU_AES_KEY_SRC_DEV :For Device Key
u32	Iv	Pointer to the Initialization Vector for decryption

Table 2: XSecure_AesInitialize Arguments (cont'd)

Type	Name	Description
u32	Key	Pointer to Aes key in case KUP key is used. Pass Null if the device key is to be used.

Returns

XST_SUCCESS if initialization was successful.

XSecure_AesDecryptInit

This function initializes the AES engine for decryption and is required to be called before calling XSecure_AesDecryptUpdate.

Note: If all of the data to be decrypted is available, the XSecure_AesDecryptData function can be used instead.

Prototype

```
u32 XSecure_AesDecryptInit(XSecure_Aes *InstancePtr, u8 *DecData, u32 Size,
u8 *GcmTagAddr);
```

Parameters

The following table lists the XSecure_AesDecryptInit function arguments.

Table 3: XSecure_AesDecryptInit Arguments

Type	Name	Description
XSecure_Aes *	InstancePtr	Pointer to the XSecure_Aes instance.
u8 *	DecData	Pointer in which decrypted data will be stored.
u32	Size	Expected size of the data in bytes whereas the number of bytes provided should be multiples of 4.
u8 *	GcmTagAddr	Pointer to the GCM tag which needs to be verified during decryption of the data.

Returns

None

XSecure_AesDecryptUpdate

This function decrypts the encrypted data passed in and updates the GCM tag from any previous calls. The size from XSecure_AesDecryptInit is decremented from the size passed into this function to determine when the GCM tag passed to XSecure_AesDecryptInit needs to be compared to the GCM tag calculated in the AES engine.

Note: When Size of the data equals to size of the remaining data that data will be treated as final data. This API can be called multiple times but sum of all Sizes should be equal to Size mention in init. Return of the final call of this API tells whether GCM tag is matching or not.

Prototype

```
s32 XSecure_AesDecryptUpdate(XSecure_Aes *InstancePtr, u8 *EncData, u32 Size);
```

Parameters

The following table lists the XSecure_AesDecryptUpdate function arguments.

Table 4: XSecure_AesDecryptUpdate Arguments

Type	Name	Description
XSecure_Aes *	InstancePtr	Pointer to the XSecure_Aes instance.
u8 *	EncData	Pointer to the encrypted data which needs to be decrypted.
u32	Size	Expected size of data to be decrypted in bytes, whereas the number of bytes should be multiples of 4.

Returns

Final call of this API returns the status of GCM tag matching.

- XSECURE_CSU_AES_GCM_TAG_MISMATCH: If GCM tag is mismatched
- XSECURE_CSU_AES_ZEROIZATION_ERROR: If GCM tag is mismatched, zeroize the decrypted data and send the status of zeroization.
- XST_SUCCESS: If GCM tag is matching.

XSecure_AesDecryptData

This function decrypts the encrypted data provided and updates the DecData buffer with decrypted data.

Note: When using this function to decrypt data that was encrypted with XSecure_AesEncryptData, the GCM tag will be stored as the last sixteen (16) bytes of data in XSecure_AesEncryptData's Dst (destination) buffer and should be used as the GcmTagAddr's pointer.

Prototype

```
s32 XSecure_AesDecryptData(XSecure_Aes *InstancePtr, u8 *DecData, u8 *EncData, u32 Size, u8 *GcmTagAddr);
```

Parameters

The following table lists the `XSecure_AesDecryptData` function arguments.

Table 5: XSecure_AesDecryptData Arguments

Type	Name	Description
XSecure_Aes *	InstancePtr	Pointer to the XSecure_Aes instance.
u8 *	DecData	Pointer to a buffer in which decrypted data will be stored.
u8 *	EncData	Pointer to the encrypted data which needs to be decrypted.
u32	Size	Size of data to be decrypted in bytes, whereas the number of bytes should be multiples of 4.

Returns

This API returns the status of GCM tag matching.

- `XSECURE_CSU_AES_GCM_TAG_MISMATCH`: If GCM tag was mismatched
- `XST_SUCCESS`: If GCM tag was matched.

XSecure_AesDecrypt

This function will handle the AES-GCM Decryption.

Note: This function is used for decrypting the Image's partition encrypted by Bootgen

Prototype

```
s32 XSecure_AesDecrypt(XSecure_Aes *InstancePtr, u8 *Dst, const u8 *Src, u32 Length);
```

Parameters

The following table lists the `XSecure_AesDecrypt` function arguments.

Table 6: XSecure_AesDecrypt Arguments

Type	Name	Description
XSecure_Aes *	InstancePtr	Pointer to the XSecure_Aes instance.
const u8 *	Src	Pointer to encrypted data source location
u8 *	Dst	Pointer to location where decrypted data will be written.
u32	Length	Expected total length of decrypted image expected.

Returns

returns XST_SUCCESS if successful, or the relevant errorcode.

XSecure_AesEncryptInit

This function is used to initialize the AES engine for encryption.

Note: If all of the data to be encrypted is available, the XSecure_AesEncryptData function can be used instead.

Prototype

```
u32 XSecure_AesEncryptInit(XSecure_Aes *InstancePtr, u8 *EncData, u32 Size);
```

Parameters

The following table lists the XSecure_AesEncryptInit function arguments.

Table 7: XSecure_AesEncryptInit Arguments

Type	Name	Description
XSecure_Aes *	InstancePtr	Pointer to the XSecure_Aes instance.
u8 *	EncData	Pointer of a buffer in which encrypted data along with GCM TAG will be stored. Buffer size should be Size of data plus 16 bytes.
u32	Size	A 32 bit variable, which holds the size of the input data to be encrypted in bytes, whereas number of bytes provided should be multiples of 4.

Returns

None

XSecure_AesEncryptUpdate

This function encrypts the clear-text data passed in and updates the GCM tag from any previous calls. The size from XSecure_AesEncryptInit is decremented from the size passed into this function to determine when the final CSU DMA transfer of data to the AES-GCM cryptographic core.

Note: If all of the data to be encrypted is available, the XSecure_AesEncryptData function can be used instead.

Prototype

```
u32 XSecure_AesEncryptUpdate(XSecure_Aes *InstancePtr, const u8 *Data, u32 Size);
```

Parameters

The following table lists the `XSecure_AesEncryptUpdate` function arguments.

Table 8: XSecure_AesEncryptUpdate Arguments

Type	Name	Description
XSecure_Aes *	InstancePtr	Pointer to the XSecure_Aes instance.
const u8 *	Data	Pointer to the data for which encryption should be performed.
u32	Size	A 32 bit variable, which holds the size of the input data in bytes, whereas the number of bytes provided should be multiples of 4.

Returns

None

XSecure_AesEncryptData

This function encrypts Len (length) number of bytes of the passed in Src (source) buffer and stores the encrypted data along with its associated 16 byte tag in the Dst (destination) buffer.

Note: If data to be encrypted is not available in one buffer one can call `XSecure_AesEncryptInit()` and update the AES engine with data to be encrypted by calling `XSecure_AesEncryptUpdate()` API multiple times as required.

Prototype

```
u32 XSecure_AesEncryptData(XSecure_Aes *InstancePtr, u8 *Dst, const u8 *Src, u32 Len);
```

Parameters

The following table lists the `XSecure_AesEncryptData` function arguments.

Table 9: XSecure_AesEncryptData Arguments

Type	Name	Description
XSecure_Aes *	InstancePtr	A pointer to the XSecure_Aes instance.
u8 *	Dst	A pointer to a buffer where encrypted data along with GCM tag will be stored. The Size of buffer provided should be Size of the data plus 16 bytes
const u8 *	Src	A pointer to input data for encryption.
u32	Len	Size of input data in bytes, whereas the number of bytes provided should be multiples of 4.

Returns

None

XSecure_AesReset

This function sets and then clears the AES-GCM's reset line.

Prototype

```
void XSecure_AesReset(XSecure_Aes *InstancePtr);
```

Parameters

The following table lists the XSecure_AesReset function arguments.

Table 10: XSecure_AesReset Arguments

Type	Name	Description
XSecure_Aes *	InstancePtr	is a pointer to the XSecure_Aes instance.

Returns

None

AES-GCM Error Codes

The table below lists the AES-GCM error codes.

Error Code	Error Value	Description
XSECURE_CSU_AES_GCM_TAG_MISMATCH	0x1	User provided GCM tag does not match with GCM calculated on data
XSECURE_CSU_AES_IMAGE_LENGTH_MISMATCH	0x2	When there is a Image length mismatch
XSECURE_CSU_AES_DEVICE_COPY_ERROR	0x3	When there is device copy error.
XSECURE_CSU_AES_ZEROIZATION_ERROR	0x4	When there is an error with Zeroization. Note: In case of any error during Aes decryption, we perform zeroization of the decrypted data.
XSECURE_CSU_AES_KEY_CLEAR_ERROR	0x20	Error when clearing key storage registers after Aes operation.

AES-GCM API Example Usage

The following example illustrates the usage of AES encryption and decryption APIs.

```
static s32 SecureAesExample(void)
{
    XCsuDma_Config *Config;
    s32 Status;
    u32 Index;
    XCsuDma_CsuDmaInstance;
    XSecure_Aes Secure_Aes;

    /* Initialize CSU DMA driver */
    Config = XCsuDma_LookupConfig(XSECURE_CSUDMA_DEVICEID);
    if (NULL == Config) {
        return XST_FAILURE;
    }

    Status = XCsuDma_CfgInitialize(&CsuDmaInstance, Config,
                                   Config->BaseAddress);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /* Initialize the Aes driver so that it's ready to use */
    XSecure_AesInitialize(&Secure_Aes, &CsuDmaInstance,
                          XSECURE_CSU_AES_KEY_SRC_KUP,
                          (u32 *)Iv, (u32 *)Key);

    xil_printf("Data to be encrypted: \n\r");
    for (Index = 0; Index < XSECURE_DATA_SIZE; Index++) {
        xil_printf("%02x", Data[Index]);
    }
    xil_printf( "\r\n\n");

    /* Encryption of Data */
    /*
     * If all the data to be encrypted is contiguous one can call
     * XSecure_AesEncryptData API directly.
     */
    XSecure_AesEncryptInit(&Secure_Aes, EncData, XSECURE_DATA_SIZE);
    XSecure_AesEncryptUpdate(&Secure_Aes, Data, XSECURE_DATA_SIZE);

    xil_printf("Encrypted data: \n\r");
    for (Index = 0; Index < XSECURE_DATA_SIZE; Index++) {
        xil_printf("%02x", EncData[Index]);
    }
    xil_printf( "\r\n");

    xil_printf("GCM tag: \n\r");
    for (Index = 0; Index < XSECURE_SECURE_GCM_TAG_SIZE; Index++) {
        xil_printf("%02x", EncData[XSECURE_DATA_SIZE + Index]);
    }
    xil_printf( "\r\n\n");

    /* Decrypt's the encrypted data */
    /*
     * If data to be decrypted is contiguous one can also call
     * single API XSecure_AesDecryptData
     */
}
```

```

XSecure_AesDecryptInit(&Secure_Aes, DecData, XSECURE_DATA_SIZE,
                      EncData + XSECURE_DATA_SIZE);
/* Only the last update will return the GCM TAG matching status */
Status = XSecure_AesDecryptUpdate(&Secure_Aes, EncData,
                                XSECURE_DATA_SIZE);
if (Status != XST_SUCCESS) {
    xil_printf("Decryption failure- GCM tag was not matched\n
\r");
    return Status;
}

xil_printf("Decrypted data\n\r");
for (Index = 0; Index < XSECURE_DATA_SIZE; Index++) {
    xil_printf("%02x", DecData[Index]);
}
xil_printf( "\r\n");

/* Comparison of Decrypted Data with original data */
for(Index = 0; Index < XSECURE_DATA_SIZE; Index++) {
    if (Data[Index] != DecData[Index]) {
        xil_printf("Failure during comparison of the data\n
\r");
        return XST_FAILURE;
    }
}

return XST_SUCCESS;
}

```

Note: Relevant examples are available in the <library-install-path>\examples folder. Where <library-install-path> is the XilSecure library installation path.

AES-GCM Usage to decrypt Boot Image

The Multiple key(Key Rolling) or Single key encrypted images will have the same format. The images include:

- Secure header - This includes the dummy AES key of 32byte + Block 0 IV of 12byte + DLC for Block 0 of 4byte + GCM tag of 16byte(Un-Enc).
- Block N - This includes the boot image data for the block N of n size + Block N+1 AES key of 32byte + Block N+1 IV of 12byte + GCM tag for Block N of 16byte(Un-Enc).

The Secure header and Block 0 will be decrypted using the device key or user provided key. If more than one block is found then the key and the IV obtained from previous block will be used for decryption.

Following are the instructions to decrypt an image:

1. Read the first 64 bytes and decrypt 48 bytes using the selected Device key.
2. Decrypt Block 0 using the IV + Size and the selected Device key.

3. After decryption, you will get the decrypted data+KEY+IV+Block Size. Store the KEY/IV into KUP/IV registers.
4. Using Block size, IV and the next Block key information, start decrypting the next block.
5. If the current image size is greater than the total image length, perform the next step. Else, go back to the previous step.
6. If there are failures, an error code is returned. Else, the decryption is successful.

RSA

The `xsecure_rsa.h` file contains hardware interface related information for the RSA hardware accelerator. This hardened cryptographic accelerator, within the CSU, performs the modulus math based on the Rivest-Shamir-Adelman (RSA) algorithm. It is an asymmetric algorithm.

Initialization & Configuration

The RSA driver instance can be initialized by using the `XSecure_RsaInitialize()` function. The method used for RSA implementation can take a pre-calculated value of $R^2 \bmod N$. If you do not have the pre-calculated exponential value pass NULL, the controller will take care of the exponential value.

Note:

- From the RSA key modulus, the exponent should be extracted.
- For verification, PKCS v1.5 padding scheme has to be applied for comparing the data hash with decrypted hash.

Table 11: Quick Function Reference

Type	Name	Arguments
s32	XSecure_RsaInitialize	XSecure_Rsa * InstancePtr u8 * Mod u8 * ModExt u8 * ModExpo
u32	XSecure_RsaSignVerification	u8 * Signature u8 * Hash u32 HashLen
s32	XSecure_RsaPublicEncrypt	XSecure_Rsa * InstancePtr u8 * Input u32 Size u8 * Result

Table 11: Quick Function Reference (cont'd)

Type	Name	Arguments
s32	XSecure_RsaPrivateDecrypt	XSecure_Rsa * InstancePtr u8 * Input u32 Size u8 * Result

Functions

XSecure_RsaInitialize

This function initializes a [XSecure_Rsa](#) structure with the default values required for operating the RSA cryptographic engine.

Note: Modulus, ModExt and ModExpo are part of prtion signature when authenticated boot image is generated by bootgen, else the all of them should be extracted from the key.

Prototype

```
s32 XSecure_RsaInitialize(XSecure_Rsa *InstancePtr, u8 *Mod, u8 *ModExt, u8 *ModExpo);
```

Parameters

The following table lists the [XSecure_RsaInitialize](#) function arguments.

Table 12: XSecure_RsaInitialize Arguments

Type	Name	Description
XSecure_Rsa *	InstancePtr	Pointer to the XSecure_Rsa instance.
u8 *	Mod	A character Pointer which contains the key Modulus of key size.
u8 *	ModExt	A Pointer to the pre-calculated exponential ($R^2 \text{ Mod } N$) value. <ul style="list-style-type: none"> NULL - if user doesn't have pre-calculated $R^2 \text{ Mod } N$ value, control will take care of this calculation internally.
u8 *	ModExpo	Pointer to the buffer which contains key exponent.

Returns

XST_SUCCESS if initialization was successful.

XSecure_RsaSignVerification

This function verifies the RSA decrypted data provided is either matching with the provided expected hash by taking care of PKCS padding.

Prototype

```
u32 XSecure_RsaSignVerification(u8 *Signature, u8 *Hash, u32 HashLen);
```

Parameters

The following table lists the `XSecure_RsaSignVerification` function arguments.

Table 13: XSecure_RsaSignVerification Arguments

Type	Name	Description
u8 *	Signature	Pointer to the buffer which holds the decrypted RSA signature
u8 *	Hash	Pointer to the buffer which has the hash calculated on the data to be authenticated.
u32	HashLen	Length of Hash used. <ul style="list-style-type: none"> For SHA3 it should be 48 bytes For SHA2 it should be 32 bytes

Returns

- `XST_SUCCESS` if decryption was successful.
- `XST_FAILURE` in case of mismatch.

XSecure_RsaPublicEncrypt

This function handles the RSA encryption with the public key components provided when initializing the RSA cryptographic core with the `XSecure_RsaInitialize` function.

Note: The Size passed here needs to match the key size used in the `XSecure_RsaInitialize` function.

Prototype

```
s32 XSecure_RsaPublicEncrypt(XSecure_Rsa *InstancePtr, u8 *Input, u32 Size, u8 *Result);
```

Parameters

The following table lists the `XSecure_RsaPublicEncrypt` function arguments.

Table 14: XSecure_RsaPublicEncrypt Arguments

Type	Name	Description
XSecure_Rsa *	InstancePtr	Pointer to the XSecure_Rsa instance.
u8 *	Input	Pointer to the buffer which contains the input data to be encrypted.
u32	Size	Key size in bytes, Input size also should be same as Key size mentioned. Inputs supported are <ul style="list-style-type: none"> XSECURE_RSA_4096_KEY_SIZE XSECURE_RSA_2048_KEY_SIZE XSECURE_RSA_3072_KEY_SIZE
u8 *	Result	Pointer to the buffer where resultant decrypted data to be stored .

Returns

- XST_SUCCESS if encryption was successful.
- Error code on failure

XSecure_RsaPrivateDecrypt

This function handles the RSA decryption with the private key components provided when initializing the RSA cryptographic core with the XSecure_RsaInitialize function.

Note: The Size passed in needs to match the key size used in the XSecure_RsaInitialize function..

Prototype

```
s32 XSecure_RsaPrivateDecrypt(XSecure_Rsa *InstancePtr, u8 *Input, u32
Size, u8 *Result);
```

Parameters

The following table lists the XSecure_RsaPrivateDecrypt function arguments.

Table 15: XSecure_RsaPrivateDecrypt Arguments

Type	Name	Description
XSecure_Rsa *	InstancePtr	Pointer to the XSecure_Rsa instance.
u8 *	Input	Pointer to the buffer which contains the input data to be decrypted.

Table 15: XSecure_RsaPrivateDecrypt Arguments (cont'd)

Type	Name	Description
u32	Size	Key size in bytes, Input size also should be same as Key size mentioned. Inputs supported are <ul style="list-style-type: none"> XSECURE_RSA_4096_KEY_SIZE, XSECURE_RSA_2048_KEY_SIZE XSECURE_RSA_3072_KEY_SIZE
u8 *	Result	Pointer to the buffer where resultant decrypted data to be stored .

Returns

- XST_SUCCESS if decryption was successful.
- XSECURE_RSA_DATA_VALUE_ERROR - if input data is greater than modulus.
- XST_FAILURE - on RSA operation failure.

Data Structures

XSecure_Rsa

The RSA driver instance data structure. A pointer to an instance data structure is passed around by functions to refer to a specific driver instance.

Declaration

```
typedef struct
{
    u32 BaseAddress,
    u8 * Mod,
    u8 * ModExt,
    u8 * ModExpo,
    u8 EncDec,
    u32 SizeInWords,
    XSecure_RsaState RsaState
} XSecure_Rsa;
```

Table 16: Structure XSecure_Rsa member description

Member	Description
BaseAddress	Device Base Address
Mod	Modulus
ModExt	Precalc. R sq. mod N
ModExpo	Exponent

Table 16: Structure XSecure_Rsa member description (cont'd)

Member	Description
EncDec	0 for signature verification and 1 for generation
SizeInWords	
RsaState	RSA key size in words

RSA API Example Usage

The following example illustrates the usage of the RSA library to encrypt data using the public key and to decrypt the data using private key.

Note: Application should take care of the padding.

```
u32 SecureRsaExample(void)
{
    u32 Index;

    /* RSA signature decrypt with private key */
    /*
     * Initialize the Rsa driver with private key components
     * so that it's ready to use
     */
    XSecure_RsaInitialize(&Secure_Rsa, Modulus, NULL, PrivateExp);

    if(XST_SUCCESS != XSecure_RsaPrivateDecrypt(&Secure_Rsa, Data,
                                                Size, Signature)) {
        xil_printf("Failed at RSA signature decryption\n\r");
        return XST_FAILURE;
    }

    xil_printf("\r\n Decrypted Signature with private key\r\n ");

    for(Index = 0; Index < Size; Index++) {
        xil_printf(" %02x ", Signature[Index]);
    }
    xil_printf(" \r\n");

    /* Verification if Data is expected */
    for(Index = 0; Index < Size; Index++) {
        if (Signature[Index] != ExpectedSign[Index]) {
            xil_printf("\r\nError at verification of RSA
signature"
                    " Decryption\n\r");
            return XST_FAILURE;
        }
    }

    /* RSA signature encrypt with Public key components */
    /*
     * Initialize the Rsa driver with public key components
     * so that it's ready to use
     */
}
```

```

        XSecure_RsaInitialize(&Secure_Rsa, Modulus, NULL, (u8 *)&PublicExp);

        if(XST_SUCCESS != XSecure_RsaPublicEncrypt(&Secure_Rsa, Signature,
                                                    Size,
EncryptSignatureOut)) {
            xil_printf("\r\nFailed at RSA signature encryption\n\r");
            return XST_FAILURE;
        }
        xil_printf("\r\n Encrypted Signature with public key\r\n ");

        for(Index = 0; Index < Size; Index++) {
            xil_printf(" %02x ", EncryptSignatureOut[Index]);
        }

        /* Verification if Data is expected */
        for(Index = 0; Index < Size; Index++) {
            if (EncryptSignatureOut[Index] != Data[Index]) {
                xil_printf("\r\nError at verification of RSA
signature"
                           " encryption\n\r");
                return XST_FAILURE;
            }
        }

        return XST_SUCCESS;
    }
}

```

Note: Relevant examples are available in the <library-install-path>\examples folder. Where <library-install-path> is the XilSecure library installation path.

SHA-3

This block uses the NIST-approved SHA-3 algorithm to generate a 384-bit hash on the input data. Because the SHA-3 hardware only accepts 104 byte blocks as the minimum input size, the input data will be padded with user selectable Keccak or NIST SHA-3 padding and is handled internally in the SHA-3 library.

Initialization & Configuration

The SHA-3 driver instance can be initialized using the `XSecure_Sha3Initialize()` function. A pointer to CsuDma instance has to be passed during initialization as the CSU DMA will be used for data transfers to the SHA module. When all the data is available on which the SHA3 hash must be calculated, the `XSecure_Sha3Digest()` can be used with the appropriate parameters as described. When all the data is not available, use the SHA3 functions in the following order:

1. `XSecure_Sha3Start()`
2. `XSecure_Sha3Update()` - This function can be called multiple times until all input data has been passed to the SHA-3 cryptographic core.
3. `XSecure_Sha3Finish()` - Provides the final hash of the data. To get intermediate hash values after each `XSecure_Sha3Update()`, you can call `XSecure_Sha3_ReadHash()` after the `XSecure_Sha3Update()` call.

Table 17: Quick Function Reference

Type	Name	Arguments
s32	<code>XSecure_Sha3Initialize</code>	<code>XSecure_Sha3 * InstancePtr</code> <code>XCsuDma * CsuDmaPtr</code>
void	<code>XSecure_Sha3Start</code>	<code>XSecure_Sha3 * InstancePtr</code>
u32	<code>XSecure_Sha3Update</code>	<code>XSecure_Sha3 * InstancePtr</code> <code>const u8 * Data</code> <code>const u32 Size</code>
u32	<code>XSecure_Sha3Finish</code>	<code>XSecure_Sha3 * InstancePtr</code> <code>u8 * Hash</code>

Table 17: Quick Function Reference (cont'd)

Type	Name	Arguments
u32	XSecure_Sha3Digest	XSecure_Sha3 * InstancePtr const u8 * In const u32 Size u8 * Out
void	XSecure_Sha3_ReadHash	XSecure_Sha3 * InstancePtr u8 * Hash
s32	XSecure_Sha3PadSelection	XSecure_Sha3 * InstancePtr Sha3Type
s32	XSecure_Sha3LastUpdate	XSecure_Sha3 * InstancePtr
u32	XSecure_Sha3WaitForDone	XSecure_Sha3 * InstancePtr

Functions

XSecure_Sha3Initialize

This function initializes a XSecure_Sha3 structure with the default values required for operating the SHA3 cryptographic engine.

Note: The base address is initialized directly with value from xsecure_hw.h The default is NIST SHA3 padding, to change to KECCAK padding call [XSecure_Sha3PadSelection\(\)](#) after XSecure_Sha3Initialize().

Prototype

```
s32 XSecure_Sha3Initialize(XSecure_Sha3 *InstancePtr, XCsuDma *CsuDmaPtr);
```

Parameters

The following table lists the XSecure_Sha3Initialize function arguments.

Table 18: XSecure_Sha3Initialize Arguments

Type	Name	Description
XSecure_Sha3 *	InstancePtr	Pointer to the XSecure_Sha3 instance.
XCsuDma *	CsuDmaPtr	Pointer to the XCsuDma instance.

Returns

XST_SUCCESS if initialization was successful

XSecure_Sha3Start

This function configures Secure Stream Switch and starts the SHA-3 engine.

Prototype

```
void XSecure_Sha3Start(XSecure_Sha3 *InstancePtr);
```

Parameters

The following table lists the XSecure_Sha3Start function arguments.

Table 19: XSecure_Sha3Start Arguments

Type	Name	Description
XSecure_Sha3 *	InstancePtr	Pointer to the XSecure_Sha3 instance.

Returns

None

XSecure_Sha3Update

This function updates the SHA3 engine with the input data.

Prototype

```
u32 XSecure_Sha3Update(XSecure_Sha3 *InstancePtr, const u8 *Data, const u32 Size);
```

Parameters

The following table lists the XSecure_Sha3Update function arguments.

Table 20: XSecure_Sha3Update Arguments

Type	Name	Description
XSecure_Sha3 *	InstancePtr	Pointer to the XSecure_Sha3 instance.
const u8 *	Data	Pointer to the input data for hashing.
const u32	Size	Size of the input data in bytes.

Returns

XST_SUCCESS if the update is successful XST_FAILURE if there is a failure in SSS config

XSecure_Sha3Finish

This function updates SHA3 engine with final data which includes SHA3 padding and reads final hash on complete data.

Prototype

```
u32 XSecure_Sha3Finish(XSecure_Sha3 *InstancePtr, u8 *Hash);
```

Parameters

The following table lists the XSecure_Sha3Finish function arguments.

Table 21: XSecure_Sha3Finish Arguments

Type	Name	Description
XSecure_Sha3 *	InstancePtr	Pointer to the XSecure_Sha3 instance.
u8 *	Hash	Pointer to location where resulting hash will be written

Returns

XST_SUCCESS if finished without any errors XST_FAILURE if Sha3PadType is other than KECCAK or NIST

XSecure_Sha3Digest

This function calculates the SHA-3 digest on the given input data.

Prototype

```
u32 XSecure_Sha3Digest(XSecure_Sha3 *InstancePtr, const u8 *In, const u32 Size, u8 *Out);
```

Parameters

The following table lists the XSecure_Sha3Digest function arguments.

Table 22: XSecure_Sha3Digest Arguments

Type	Name	Description
XSecure_Sha3 *	InstancePtr	Pointer to the XSecure_Sha3 instance.
const u8 *	In	Pointer to the input data for hashing

Table 22: XSecure_Sha3Digest Arguments (cont'd)

Type	Name	Description
const u32	Size	Size of the input data
u8 *	Out	Pointer to location where resulting hash will be written.

Returns

XST_SUCCESS if digest calculation done successfully XST_FAILURE if any error from Sha3Update or Sha3Finish.

XSecure_Sha3_ReadHash

This function reads the SHA3 hash of the data and it can be called between calls to XSecure_Sha3Update.

Prototype

```
void XSecure_Sha3_ReadHash(XSecure_Sha3 *InstancePtr, u8 *Hash);
```

Parameters

The following table lists the XSecure_Sha3_ReadHash function arguments.

Table 23: XSecure_Sha3_ReadHash Arguments

Type	Name	Description
XSecure_Sha3 *	InstancePtr	Pointer to the XSecure_Sha3 instance.
u8 *	Hash	Pointer to a buffer in which read hash will be stored.

Returns

None

XSecure_Sha3PadSelection

This function provides an option to select the SHA-3 padding type to be used while calculating the hash.

Note: The default provides support for NIST SHA-3. If a user wants to change the padding to Keccak SHA-3, this function should be called after `XSecure_Sha3Initialize()`

Prototype

```
s32 XSecure_Sha3PadSelection(XSecure_Sha3 *InstancePtr, XSecure_Sha3PadType Sha3PadType);
```

Parameters

The following table lists the `XSecure_Sha3PadSelection` function arguments.

Table 24: XSecure_Sha3PadSelection Arguments

Type	Name	Description
XSecure_Sha3 *	InstancePtr	Pointer to the XSecure_Sha3 instance.
Commented parameter Sha3Type does not exist in function XSecure_Sha3PadSelection.	Sha3Type	Type of SHA3 padding to be used. <ul style="list-style-type: none"> For NIST SHA-3 padding - XSECURE_CSU_NIST_SHA3 For KECCAK SHA-3 padding - XSECURE_CSU_KECCAK_SHA3

Returns

XST_SUCCESS if pad selection is successful. XST_FAILURE if pad selection is failed.

XSecure_Sha3LastUpdate

This function is to notify this is the last update of data where sha padding is also been included along with the data in the next update call.

Prototype

```
s32 XSecure_Sha3LastUpdate(XSecure_Sha3 *InstancePtr);
```

Parameters

The following table lists the `XSecure_Sha3LastUpdate` function arguments.

Table 25: XSecure_Sha3LastUpdate Arguments

Type	Name	Description
XSecure_Sha3 *	InstancePtr	Pointer to the XSecure_Sha3 instance.

Returns

XST_SUCCESS if last update can be accepted

XSecure_Sha3WaitForDone

This inline function waits till SHA3 completes its operation.

Prototype

```
u32 XSecure_Sha3WaitForDone(XSecure_Sha3 *InstancePtr);
```

Parameters

The following table lists the XSecure_Sha3WaitForDone function arguments.

Table 26: XSecure_Sha3WaitForDone Arguments

Type	Name	Description
XSecure_Sha3 *	InstancePtr	Pointer to the XSecure_Sha3 instance.

Returns

XST_SUCCESS if the SHA3 completes its operation. XST_FAILURE if a timeout has occurred.

SHA-3 API Example Usage

The xilsecure_sha_example.c file is a simple example application that demonstrates the usage of SHA-3 accelerator to calculate a 384-bit hash on the Hello World string. A typical use case for the SHA3 accelerator is for calculation of the boot image hash as part of the authentication operation. This is illustrated in the xilsecure_rsa_example.c.

The contents of the xilsecure_sha_example.c file are shown below:

```
int SecureHelloWorldExample()
{
    u8 HelloWorld[4] = {'h','e','l','l'};
    u32 Size = sizeof(HelloWorld);
    u8 Out[384/8];
    XCsuDma_Config *Config;

    int Status;

    Config = XCsuDma_LookupConfig(0);
    if (NULL == Config) {
        xil_printf("config failed\n\r");
        return XST_FAILURE;
    }

    Status = XCsuDma_CfgInitialize(&CsuDma, Config, Config->BaseAddress);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Initialize the SHA-3 driver so that it's ready to use
     */
    XSecure_Sha3Initialize(&Secure_Sha3, &CsuDma);
```

```
XSecure_Sha3Digest(&Secure_Sha3, HelloWorld, Size, Out);

xil_printf(" Calculated Digest \r\n ");
int i= 0;
for(i=0; i< (384/8); i++)
{
    xil_printf(" %0x ", Out[i]);
}
xil_printf(" \r\n ");

return XST_SUCCESS;
}
```

Note: The `xilsecure_sha_example.c` and `xilsecure_rsa_example.c` example files are available in the <library-install-path>\examples folder. Where <library-install-path> is the XilSecure library installation path.

XilSecure Utilities

The `xsecure_utils.h` file contains common functions used among the XilSecure library like holding hardware crypto engines in Reset or bringing them out of reset, and secure stream switch configuration for AES and SHA3.

For more information on the Linux interface to Zynq UltraScale+ MPSoC device secure modules, see:

- [Zynq Ultrascale+ MPSoC Secure Driver for Linux](#)
- [RSA Driver](#)
- [Linux SHA Driver for Zynq Ultrascale+ MPSoC](#)
- [AES Driver](#)

For more information on the U-boot interface to Zynq UltraScale+ MPSoC device secure modules, refer [Security Features Usage in U-boot](#).

Table 27: Quick Function Reference

Type	Name	Arguments
u32	XSecure_ReadReg	u32 BaseAddress u16 RegOffset
void	XSecure_WriteReg	u32 BaseAddress u32 RegOffset u32 RegisterValue
void	XSecure_SetReset	u32 BaseAddress u32 BaseAddress
void	XSecure_ReleaseReset	u32 BaseAddress u32 BaseAddress

Functions

XSecure_ReadReg

Read from the register.

Note: C-Style signature: u32 XSecure_ReadReg(u32 BaseAddress, u16 RegOffset)

Prototype

```
u32 XSecure_ReadReg(u32 BaseAddress, u16 RegOffset);
```

Parameters

The following table lists the XSecure_ReadReg function arguments.

Table 28: XSecure_ReadReg Arguments

Type	Name	Description
u32	BaseAddress	contains the base address of the device.
u16	RegOffset	contains the offset from the base address of the device.

Returns

The value read from the register.

XSecure_WriteReg

Write to the register.

Note: C-Style signature: void XSecure_WriteReg(u32 BaseAddress, u16 RegOffset, u16 RegisterValue)

Prototype

```
void XSecure_WriteReg(u32 BaseAddress, u32 RegOffset, u32 RegisterValue);
```

Parameters

The following table lists the XSecure_WriteReg function arguments.

Table 29: XSecure_WriteReg Arguments

Type	Name	Description
u32	BaseAddress	contains the base address of the device.

Table 29: XSecure_WriteReg Arguments (cont'd)

Type	Name	Description
u32	RegOffset	contains the offset from the base address of the device.
u32	RegisterValue	is the value to be written to the register

Returns

None.

XSecure_SetReset

This function places the hardware core into the reset.

Prototype

```
void XSecure_SetReset(u32 BaseAddress, u32 Offset);
```

Parameters

The following table lists the XSecure_SetReset function arguments.

Table 30: XSecure_SetReset Arguments

Type	Name	Description
u32	BaseAddress	Base address of the core.
u32	BaseAddress	Offset of the reset register.

Returns

None

XSecure_ReleaseReset

This function takes the hardware core out of reset.

Prototype

```
void XSecure_ReleaseReset(u32 BaseAddress, u32 Offset);
```

Parameters

The following table lists the XSecure_ReleaseReset function arguments.

Table 31: XSecure_ReleaseReset Arguments

Type	Name	Description
u32	BaseAddress	Base address of the core.
u32	BaseAddress	Offset of the reset register.

Returns

None

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

Documentation Navigator and Design Hubs

Xilinx[®] Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado[®] IDE, select **Help** → **Documentation and Tutorials**.
- On Windows, select **Start** → **All Programs** → **Xilinx Design Tools** → **DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

Note: For more information on DocNav, see the [Documentation Navigator](#) page on the Xilinx website.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2020 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.