

## Summary

The LibXilSecure library provides APIs to access secure hardware on the Zynq® UltraScale+™ MPSoC devices.

This library includes:

- SHA-3 engine hash functions
- AES for symmetric key encryption
- RSA for authentication

These are grouped into the Configuration and Security Unit (CSU) on the Zynq UltraScale+ MPSoC device.

## XilSecure APIs

The following list is a list of functions for Zynq® UltraScale+™ MPSoC devices which are grouped by their respective function. You can click on a link to go directly to the function section.

### Source Files

- `xsecure_hw.h`: This file contains the hardware interface for all the three modules.
- `xsecure_sha.h`: This file contains the driver interface for SHA-3 module.
- `xsecure_sha.c`: This file contains the implementation of the driver interface for SHA-3 module.
- `xsecure_rsa.h`: This file contains the driver interface for RSA module.
- `xsecure_rsa.c`: This file contains the implementation of the driver interface for RSA module.
- `xsecure_aes.h`: This file contains the driver interface for AES module.
- `xsecure_aes.c`: This file contains the implementation of the driver interface for AES module.

### SHA-3 Functions

This block uses the NIST-approved SHA-3 algorithm to generate 384 bit hash on the input data. Because the SHA-3 hardware only accepts 104 byte blocks as minimum input size, the input data is padded with a 10\*1 sequence to complete the final byte block. The padding is handled internally by the driver API.

### API Summary

The following is a summary list of APIs provided for using SHA-3 module. Descriptions of the APIs follow the list.

```
s32 XSecure_Sha3Initialize(XSecure_Sha3 *InstancePtr, XCsuDma* CsuDmaPtr)
void XSecure_Sha3Update(XSecure_Sha3 *InstancePtr, const u8 *Data, const u32 Size)
void XSecure_Sha3Start(XSecure_Sha3 *InstancePtr)
void XSecure_Sha3Finish(XSecure_Sha3 *InstancePtr, u8 *Hash)
void XSecure_Sha3Digest(XSecure_Sha3 *InstancePtr, const u8 *In, const u32 Size u8 *Out)
```

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

<b>Description</b>	This API initializes a specific Xsecure_Sha3 instance so that it is ready to be used.
<b>Parameters</b>	InstancePtr is a pointer to the XSecure_Sha3 instance. CsuDmaPtr is the pointer to the XCsuDma instance.
<b>Returns</b>	XST_SUCCESS if initialization was successful.

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

<b>Description</b>	This API configures the secure stream switch (SSS) and starts the SHA-3 engine.
<b>Parameters</b>	InstancePtr is a pointer to the XSecure_Sha3 instance.
<b>Returns</b>	None

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

<b>Description</b>	This API updates hash for new input data block. This is used after XSecure_Sha3Start to update more input data. InstancePtr is a pointer to the XSecure_Sha3 instance.
<b>Parameters</b>	Data is the pointer to the input data for hashing Size of the input data in bytes
<b>Returns</b>	None

```
void XSecure_Sha3Finish(XSecure_Sha3 *InstancePtr, u8
    *Hash)
```

<b>Description</b>	This API sends the last block; for example, the padding when block size is not multiple of 104 bytes. The final hash is ready at this stage.
<b>Parameters</b>	InstancePtr is a pointer to the XSecure_Sha3 instance. Hash is the pointer to location where resulting hash will be written
<b>Returns</b>	None

---

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

<b>Description</b>	This API calculates SHA-3 Digest on the given input data. In effect, this does the complete operation that can be achieved by calling XSecure_Sha3Start, XSecure_Sha3Update, and XSecure_Sha3Finish in succession.
<b>Parameters</b>	<p>InstancePtr is a pointer to the XSecure_Sha3 instance.</p> <p>In is the pointer to the input data for hashing size of the input data in bytes.</p> <p>Out is the pointer to location where resulting hash is written.</p>
<b>Returns</b>	None

### Example Usage

XSecure\_Sha3Example.c : This example is a simple application using SHA-3 device to calculate 384 bit hash on Hello World string.

A more typical use case of SHA-3 has been illustrated in RSA example XSecure\_RsaExample.c where it is used to calculate hash of boot image as a step in authentication process.

## RSA

This block decrypts data based on RSA-4096 algorithm. A utility function to compare the signature with expected signature (Verification) is also provided.

### API Summary

The following is a summary list of APIs provided for using RSA module.. Descriptions of the APIs follow the list.

```
s32 XSecure_RsaInitialize(XSecure_Rsa *InstancePtr, u8 *Mod, u8 *ModExt, u8 *ModExpo)
s32 XSecure_RsaDecrypt(XSecure_Rsa *InstancePtr, u8 *EncText, u8 *Result)
u32 XSecure_RsaSignVerification(u8 *Signature, u8 *Hash, u32 HashLen)
```

---

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

<b>Description</b>	This API initializes a specific Xsecure_Rsa instance so that it is ready to be used.
<b>Parameters</b>	<p>InstancePtr is a pointer to the XSecure_Rsa instance.</p> <p>Mod is the pointer to Modulus used for authentication.</p> <p>ModExt is the pointer to precalculated R<sup>2</sup> Mod N value used for authentication.</p> <p>ModExpo is the pointer to the exponent (public key) used for authentication.</p>
<b>Returns</b>	XST_SUCCESS if decryption was successful.

```
s32 XSecure_RsaDecrypt(XSecure_Rsa *InstancePtr, u8
    *EncText, u8 *Result)
```

**Description** This API handles the RSA decryption from end-to-end.  
InstancePtr is a pointer to the XSecure\_Rsa instance.

**Parameters** Result is the pointer to decrypted data generated by RSA.  
EncText is the pointer to the data (hash) to be decrypted.

**Returns** XST\_SUCCESS if decryption was successful.

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

**Description** This API matches the decrypted data with expected data.  
InstancePtr is a pointer to the XSecure\_Rsa instance.

**Parameters** Signature is the pointer to RSA signature for data to be authenticated.  
Hash is the pointer to expected hash data.  
HashLen is the length of Hash used.

**Returns** XST\_SUCCESS if decryption was successful.

### Example Usage

XSecure\_RsaExample.c : This example deals with RSA based authentication of FSBL in a Zynq MPSoC boot image. The boot image signature is decrypted using RSA- 4096 algorithm. Resulting digest is matched with SHA digest calculated on the FSBL using SHA-3 driver.

The authenticated boot image should be loaded in memory through JTAG and address of the boot image should be passed to the function. By default, the example assumes that the authenticated image is present at location 0x04000000 (DDR), which can be changed as required.

## AES

This block can encrypt/decrypt data using AES-GCM algorithm. Decryption using keyrolling is also supported. The key, IV and the format of the encrypted data should be the same as the one used by Bootgen for encrypting Zynq UltraScale + MPSoC device boot images. The bootgen encrypted images have a secure header at the beginning followed by any number of blocks.

Decryption in chunks for bitstreams is supported in cases where the entire bitstream is not present in single contiguous location. For example, the DDR less systems.

### API Summary

The following is a summary list of APIs provided for using AES module. Descriptions of the APIs follow the list.

```
s32 XSecure_AesInitialize(XSecure_Aes *InstancePtr, XCsuDma *CsuDmaPtr, u32 KeySel,
    u32* Iv, u32* Key)
```

```
void XSecure_AesSetChunking(XSecure_Aes *InstancePtr, u8 Chunking)
```

```
void XSecure_AesSetChunkConfig(XSecure_Aes *InstancePtr, u8 *ReadBuffer, u32
    ChunkSize, u32(*DeviceCopy)(u32, u64, u32))
```

```
s32 XSecure_AesDecrypt(XSecure_Aes *InstancePtr, u8 *Dst, const u8 *Src, *32 Length)
```

```
void XSecure_AesEncrypt(XSecure_Aes *InstancePtr, u8 *Dst, const u8 *Src 32 Len)
```

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

```
s32 XSecure_AesInitialize(XSecure_Aes *InstancePtr,
    XCsuDma *CsuDmaPtr, u32 KeySel, u32* Iv, u32* Key)
```

**Description** This API initializes the instance pointer.

**Parameters** InstancePtr is a pointer to the XSecure\_Aes instance.  
 CsuDmaPtr is the pointer to the XCsuDma instance.  
 KeySel is the key source for decryption, can be KUP (user- provided) or device key.  
 Iv is pointer to the Initialization Vector for decryption.  
 Key is the pointer to Aes decryption key in case KUP key is used. Passes Null if device key is to be used.

**Returns** XST\_SUCCESS upon success.

---

```
void XSecure_AesSetChunking(XSecure_Aes *InstancePtr, u8
    Chunking)
```

**Description** This function handles the configuration for bitstream chunking.

**Parameters** InstancePtr is a pointer to the XSecure\_Aes instance.  
 Chunking is used to enable or disable data chunking

**Returns** None

---

```
void XSecure_AesSetChunkConfig(XSecure_Aes *InstancePtr,
    u8 *ReadBuffer, u32 ChunkSize, u32(*DeviceCopy)(u32,
    u64, u32))
```

**Description** This function handles the AES-GCM decryption.

**Parameters** InstancePtr is a pointer to the XSecure\_Aes instance.  
 ReadBuffer is the buffer where the data will be written after copy.  
 ChunkSize is the length of the buffer in bytes.  
 DeviceCopy is the function pointer to copy data from device to buffer.  
 Return value of DeviceCopy should be 0 in case of success and 1 in case of failure. Arguments of DeviceCopy are:  
     SrcAddress: Address of data in device.  
     DestAddress: Address where data will be copied in chunks  
     Length: Length of data in bytes.

**Returns** None .

---

```
s32 XSecure_AesDecrypt(XSecure_Aes *InstancePtr, u8 *Dst,
    const u8 *Src, *32 Length)
```

<b>Description</b>	This function handles the AES-GCM decryption.
<b>Parameters</b>	<p><i>InstancePtr</i> is a pointer to the <i>XSecure_Aes</i> instance.</p> <p><i>Src</i> is the pointer to encrypted data source location</p> <p><i>Dst</i> is the pointer to location where decrypted data will be written.</p> <p><i>Length</i> is the expected total length of decrypted image/data.</p>
<b>Returns</b>	<p>XST_SUCCESS if encryption passed and GCM tag matched.</p> <p>XSECURE_CSU_AES_IMAGE_LEN_MISMATCH if Image length did not match.</p> <p>XSECURE_CSU_AES_GCM_TAG_MISMATCH if GCM tag mismatch occurs.</p> <p>XSECURE_CSU_AES_DEVICE_COPY_ERROR if copy of chunk from device failed.</p> <p>XST_FAILURE in case of failure.</p>

---

```
void XSecure_AesEncrypt(XSecure_Aes *InstancePtr, u8 *Dst,
    const u8 *Src 32 Len)
```

<b>Description</b>	<p>This API encrypts input data using encryption engine.</p> <p><i>InstancePtr</i> is a pointer to the <i>XSecure_Aes</i> instance.</p>
<b>Parameters</b>	<p><i>Dst</i> is pointer to location where encrypted output will be written.</p> <p><i>Src</i> is pointer to input data for encryption.</p> <p><i>Len</i> is the size of input data in bytes</p>
<b>Returns</b>	None

---

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

<b>Description</b>	This API resets the AES engine.
<b>Parameters</b>	<i>InstancePtr</i> is a pointer to the <i>XSecure_Aes</i> instance.
<b>Returns</b>	None

---

### Example Usage

*XSecure\_AesExample.c*: This example illustrates AES usage with decryption of a Zynq UltraScale + MPSoC boot image placed at a predefined location in memory. User can select the key type (device key or user-selected KUP key). The example assumes that the boot image is present at 0x0400000 (DDR); consequently, the image must be loaded at that address through JTAG. The example decrypts the boot image and returns XST\_SUCCESS or XST\_FAILURE based on whether the GCM tag was successfully matched.