



Integrity CAAM Driver
Documentation and User's Guide
December 2017, version 1

Table of Contents
3.12.2

wolfSSL

[1.0 INTRO](#)

[2.0 Building](#)

[2.1 Building The Driver](#)

[2.2 Building wolfSSL](#)

[3.0 Use](#)

[3.1 Initialization](#)

[3.2 Sending the Driver a Command](#)

[3.3 AES](#)

[3.3.1 AES-ECB](#)

[3.3.2 AES-CBC](#)

[3.3.3 AES-CTR](#)

[3.3.4 AES-CCM](#)

[3.4 Hash](#)

[3.5 Blobs](#)

[3.6 TRNG](#)

[3.7 Error Values](#)

[3.8 Threaded Applications](#)

[4.0 Appendix](#)

[4.1 API](#)

1.0 INTRO

This driver was created to make use of iMX.6 hardware acceleration with the Integrity OS. The driver operates from the kernel space where permission is granted to access addresses needed to operate the CAAM. Permission to access these addresses are restricted by the OS in a users application. The general flow for use is that a kernel with the driver is made, then the user's application running on the created kernel makes a request for the IODevice resource. After the application gets the IODevice it then sends commands along with buffers to be processed. [Section 3.0](#) about "Use" goes into more detail about the commands and buffers passed to the IODevice. All code for the driver is located at wolfssl-root/wolfcrypt/src/port/caam/caam_driver.c with a header file located at wolfssl-root/wolfssl/wolfcrypt/port/caam/caam_driver.h.

The driver supports:

- AES-CCM
- AES-ECB
- AES-CBC
- AES-CTR
- MD5
- SHA1
- SHA-224
- SHA-256
- TRNG
- Blob creation and opening

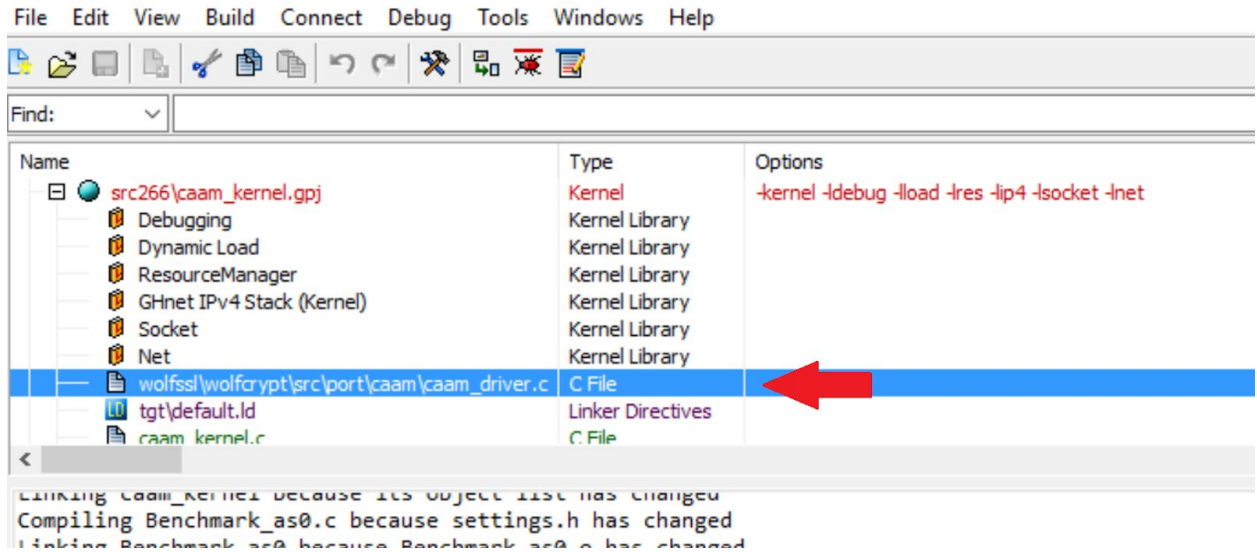
NOTE: The same logic used for SHA-256/224 can be used for SHA-384/512 and is in place but has not completely been tested due to hardware support.

2.0 Building

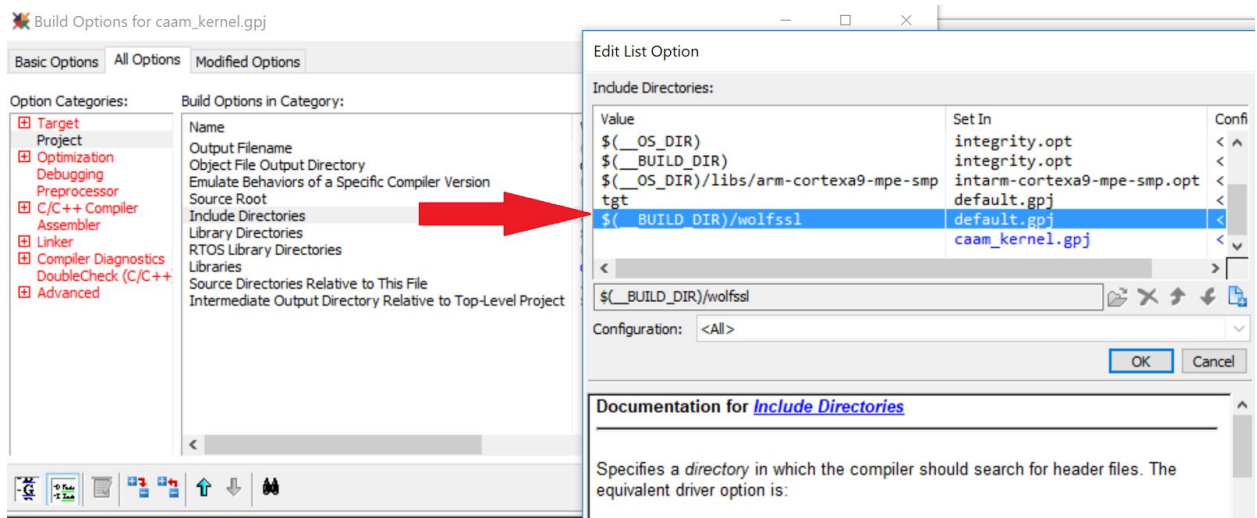
This covers two sections. First, building the kernel to incorporate the driver and second, building wolfSSL to use the driver.

2.1 Building The Driver

- To build a kernel with the driver add the file caam_driver.c to the kernel build.



- The next step is to include the path to wolfssl-root for including the file wolfssl/wolfcrypt/port/caam/caam_driver.h. This include path must be added to the project.



- Next, build the project and create a .uimage from the output. This can be done in many different ways, for the digiConnectCore6 board it was done using the elfloader.exe and arm_elfloader.exe sent with Integrity along with using mkimage. The following is an example of the commands that would be used if the kernel's name in the MULTI project was caam_kernel.

```
$ C:\ghs\comp_201516\elfloader.exe -l
C:\ghs\int1144\digiConnectCore6\arm_elfloader.bin -o
kernel.elf caam_kernel
$ mkimage -n wolfSSL -A arm -O Linux -C none -a 0x18000000 -d
kernel.elf kernel.uimage
```

- After the uimage has been created place it in the root directory of an SD card and start the board up with the SD card inserted.

```

File Edit Setup Control Window Help
Kernel.....INTEGRITY v11.4.4
BSP.....digiConnectCore6
Debug Agent.....INDRT2 v2.0.53
RAM.....1024 MB
Active Cores.....4
Initial Objects.....108
Initializing boot modules:
  Debug (INDRT).....Success
  Start Run-Mode Partner Task.....Success
  Resource Manager.....Success
  System Resources.....Success
  Driver Debug Stack.....Success
  GIPC Target.....Success
  Dynamic Loader.....Success
  SDIO/MM Card IO.....Success
  TCP/IP.....Success
  OSA Helper.....Success
  Configure I/O Termination Tasks.....Success
  Late KernelRange Clear.....Success
Boot Initialization.....Completed
TCP/IP: Driver: VirtualDebugU2
TCP/IP: FECI01          MAC: 00:40:9d:8b:87:9a  Max Speed: 100Mbps
TCP/IP: OnChip          Number: 0
TCP/IP: Interface: et0  ConfigID: 0

```

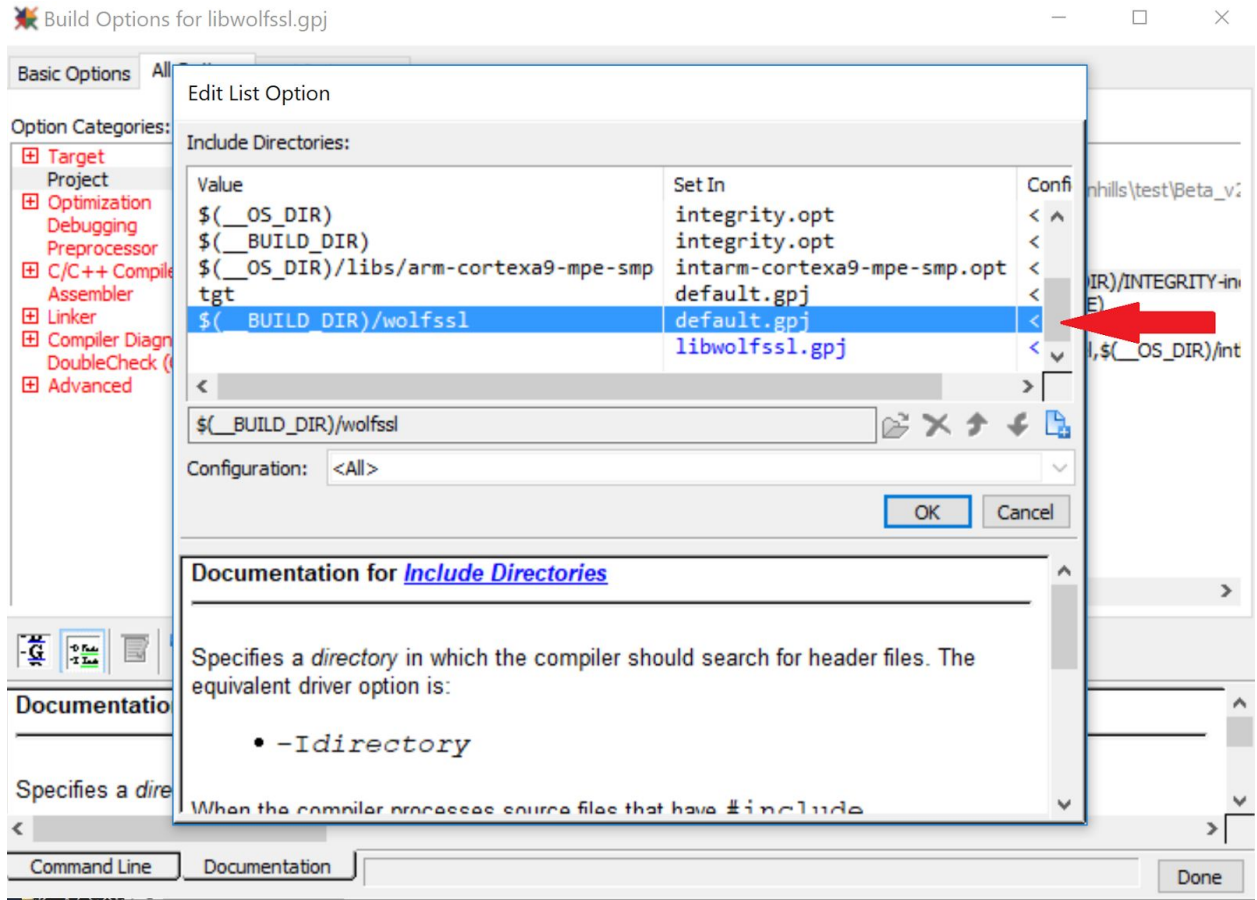
- It is now ready for an application to be loaded.

2.2 Building wolfSSL

- This section covers building wolfSSL to make use of the driver. To build wolfSSL source files from wolfssl-root/wolfcrypt/src/*.c (except misc.c if using INLINE), source files from wolfssl-root/src/*.c and source files from wolfssl-root/wolfcrypt/src/port/caam/*.c (except caam_driver.c) should be compiled.

src2\libwolfssl.gpj		Library
src262\wolfcrypt.gpj		Subproject
	wolfssl\wolfcrypt\src\port\caam\caam_aes.c	C File
	wolfssl\wolfcrypt\src\port\caam\caam_init.c	C File
	wolfssl\wolfcrypt\src\port\caam\caam_sha.c	C File
	wolfssl\wolfcrypt\src\aes.c	C File
	wolfssl\wolfcrypt\src\arc4.c	C File
	wolfssl\wolfcrypt\src\asm.c	C File
	wolfssl\wolfcrypt\src\asn.c	C File
	wolfssl\wolfcrypt\src\blake2b.c	C File
	wolfssl\wolfcrypt\src\camellia.c	C File
	wolfssl\wolfcrypt\src\chacha.c	C File
	wolfssl\wolfcrypt\src\chacha20_poly1305.c	C File
	wolfssl\wolfcrypt\src\cmac.c	C File
	wolfssl\wolfcrypt\src\coding.c	C File
	wolfssl\wolfcrypt\src\compress.c	C File
	wolfssl\wolfcrypt\src\cpuid.c	C File
	wolfssl\wolfcrypt\src\curve25519.c	C File
	wolfssl\wolfcrypt\src\des3.c	C File
	wolfssl\wolfcrypt\src\dh.c	C File
	wolfssl\wolfcrypt\src\dsa.c	C File
	wolfssl\wolfcrypt\src\ecc.c	C File
	wolfssl\wolfcrypt\src\ecc_fp.c	C File
	wolfssl\wolfcrypt\src\ed25519.c	C File
	wolfssl\wolfcrypt\src\error.c	C File
	wolfssl\wolfcrypt\src\fe_low_mem.c	C File
	wolfssl\wolfcrypt\src\fe_operations.c	C File


- A path to wolfssl-root/ must be added to the include paths for the project.



- Lastly, customization of the build is done with the use of macros. Note that if using a user_settings.h file for this the macro WOLFSSL_USER_STTINGS must be defined. The following is a list of the macros that pertain to i.MX6 builds.
 1. **WOLFSSL_IMX6** : This macro sets the size of long long to 8 if it is not already set. The macro is meant to be used for build with i.MX6 boards.
 2. **WOLFSSL_IMX6_CAAM** : This macro sets the build to use all available hardware crypto. This includes getting entropy from the TRNG, access to creating blobs, AES operations and hashing algorithms.
 3. **WOLFSSL_IMX6_CAAM_RNG** : This macro turns on using the TRNG only.
 4. **WOLFSSL_IMX6_CAAM_BLOB** : This macro turns on access to creating/opening blobs only.
 5. **WOLFSSL_IMX6_CAAM_PRINT** : Used for extra debug printing. Note that when defined stdio.h is included by the project and calls to printf are made.
 6. **NO_IMX6_CAAM_AES**: When defined the software implementation of AES operations are used.

7. **NO_IMX6_CAAM_HASH**: When defined the software implementation of hashing operations are used.
8. **WC_CAAM_PASSWORD** : Controls the password used when trying to get the IODevice on startup. It is set to the default of “!systempassword” if not defined by the user at compile time. Note that there is also the API of `wc_caamSetResource(IODevice ioDev)` to allow for run time setting of CAAM IODevice.

```
44 #define WOLFSSL_SHA3
45 #define WOLFSSL_SHA224
46
47 /* Not supported by board? Gives mode error */
48 // #define WOLFSSL_SHA384
49 // #define WOLFSSL_SHA512
50
51 /* AES operations */
52 #define WOLFSSL_AES_DIRECT
53 #define WOLFSSL_AES_COUNTER
54 #define WOLFSSL_AES_XTS
55 #define HAVE_AESCCM
56 #define HAVE_AESGCM
57
58 /* Use buffers and no file system */
59 #define USE_CERT_BUFFERS_2048
60 #define USE_CERT_BUFFERS_256
61 #define NO_FILESYSTEM
62
63 #define HAVE_CHACHA
64 #define HAVE_POLY1305
65 #define HAVE_ONE_TIME_AUTH
66
67 #define WOLFSSL_IMX6
68 #define WOLFSSL_IMX6_CAAM /* Turns on all CAAM support */
69 // #define WOLFSSL_IMX6_CAAM_RNG /* turns on use of CAAM TRNG only */
70 // #define WOLFSSL_IMX6_CAAM_BLOB /* turns on use of CAAM Blobs only */
71 // #define WOLFSSL_CAAM_PRINT /* extra print messages using printf */
72
73 #endif
```



3.0 Use

The driver can be used outside of wolfSSL API calls but it is designed and developed for use with wolfSSL. These sections cover how to use the driver and how wolfSSL uses the driver.

3.1 Initialization

Initialization of the driver is done by requesting the IODevice from the OS. A default name of “wolfSSL_CAAM_Driver” is given to the IODevice. The Integrity OS API used for requesting the IODevice is RequestResource (see Integrity documentation for more on the API). The next portion covers how wolfSSL performs initialization by making a call to RequestResource or by having the IODevice set by the user.

By default when wolfSSL_Init() or wolfCrypt_Init() is called and the macro for i.MX6 hardware crypto use is defined, wolfSSL will try to get the IODevice resource from the OS. Because the attempt to get the resource is with the default password of “!systempassword” it is not a hard failure if unable to get the resource. Instead a debug message and error will be displayed (when DEBUG_WOLFSSL macro is defined and a call to the function wolfSSL_Debugging_ON() is made). If the password set by the macro WC_CAAM_PASSWORD is unable to get the IODevice then the API wc_caamSetResource must be called with the IODevice as an argument. In the case that no IODevice is able to be set all attempts to use the hardware crypto will result in error values being returned.

3.2 Sending the Driver a Command

A command is sent to the driver by creating an array of type Buffer and an unsigned int array of length 4. Both the buffers and unsigned int array are sent to the Integrity API of SynchronousSendIORequest (more on this API can be found in the Integrity documentation). The driver is expecting the Buffers to be in a specific order and will give errors or wrong cipher/plain texts in the case that a different order is received. For example AES operations have the general buffer order of KEY -> optional IV -> Input -> Output. In the case of AES-CCM, there is the additional data buffer that is sent after the IV and with AES-ECB no optional IV is sent. Sent along with the buffers is the array of unsigned ints holding information about what is to be done with them. An example of values in the array is with AES having the first index of the array contain encrypt/decrypt flag, the second contain the key size, the third having the input/output size, and the fourth containing additional data size in the case of AES-CCM. Along with the buffer and unsigned int array, a type is sent to the driver. The type controls which algorithm is used. An example call to SynchronousSendIORequest with these three things would be the following.

```
SynchronousSendIORequest(ioDev, CAAM_AESECB, args, buffers);
```

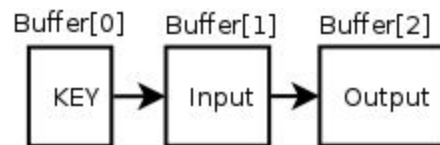
3.3 AES

This goes into depth about how wolfSSL is using the IODevice for AES operations.

3.3.1 AES-ECB

Type: CAAM_AESECB

Buffer Pattern:



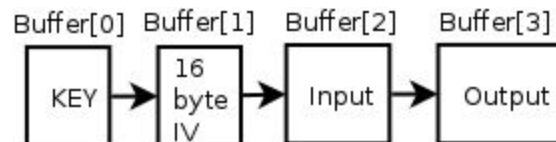
Argument Pattern:

1. CAAM_ENC / CAAM_DEC
2. Key size in bytes
3. Input / Output size in bytes

3.3.2 AES-CBC

Type: CAAM_AESCBC

Buffer Pattern:



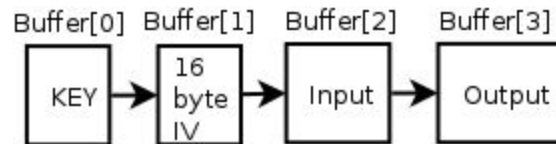
Argument Pattern:

1. CAAM_ENC / CAAM_DEC
2. Key size in bytes
3. Input / Output size in bytes

3.3.3 AES-CTR

Type: CAAM_AESCTR

Buffer Pattern:



Argument Pattern:

1. CAAM_ENC / CAAM_DEC
2. Key size in bytes
3. Input / Output size in bytes

3.3.4 AES-CCM

Type: CAAM_AESCCM

Buffer Pattern:



Argument Pattern:

1. CAAM_ENC / CAAM_DEC
2. Key size in bytes
3. Input / Output size in bytes
4. AAD size in bytes (note is unformatted)

3.4 Hash

All wolfSSL hashing API does not change when using the driver versus using the software implementation. This goes into some depth though about how wolfSSL is using the IODevice for hashing operations.

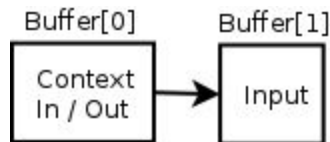
All hashing algorithms have a rolling context used to keep track of the state. The size of the context buffer must be the size of the digest plus 8 bytes. In the case of SHA224

and SHA384 it should be the size of SHA256 digest and SHA512 digest, respectively. All implemented hash algorithms follow the same pattern for expected buffers and arguments. On each call the context is written to by the driver, updating the state of the rolling digest. When CAAM_ALG_FINAL is used the context will contain the digest output.

The types passed to the IODevice can be:

- CAAM_MD5
- CAAM_SHA
- CAAM_SHA224
- CAAM_SHA256

Buffer Pattern:



Argument Pattern:

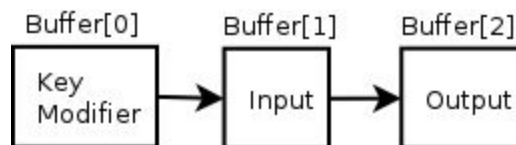
1. CAAM_ALG_INIT / CAAM_ALG_UPDATE / CAAM_ALG_FINAL
2. Context size in bytes (digest size + 8 bytes)

3.5 Blobs

Blobs are created by encrypting data using the master key from the hardware. The driver is set to use Red Key types.

Type: CAAM_BLOB_ENCAP / CAAM_BLOB_DECAP

Buffer Pattern:



Argument Pattern:

1. N/A
2. N/A

3. Input Size

3.6 TRNG

wolfSSL uses the TRNG to seed its HASH-DRBG algorithm when either the macro WOLFSSL_IMX6_CAAM or WOLFSSL_IMX6_CAAM_RNG are defined. Note that if the hardware does not have entropy ready the Integrity enum value “Waiting” is returned. The following is the type, buffers, and arguments for getting entropy from the driver.

Type: CAAM_ENTROPY

Buffer Pattern: A single output buffer.

Argument Pattern: NONE (can be NULL)

3.7 Error Values

There is a wide range of error values that can be returned from the driver. wolfSSL can print out the exact return value from the driver when in debug mode but the return values from the driver are handled as three possible states.

- WC_HW_WAIT_E (value of -249) is returned when the Integrity OS error ResourceNotAvailable is received from the driver. This error indicates that the current IOResult is not available at the moment and to try again later.
- RAN_BLOCK_E (value of -105) is returned when the Integrity OS error Waiting is received from the driver. A waiting error indicates that the current operation could not be performed and a call again should be made later. This is most common when trying to read TRNG at a time when the hardware has none available yet.
- WC_HW_E (value of -248) is returned when any error was encountered with the driver. After the driver encounters an error of this kind the current job ring is flushed and a soft reset is attempted.
- The value of 0 indicates a success.

This is a list of all values returned by the driver:

Integrity OS Enum Returned	Description
NotRestartable	Is returned if a soft reset fails. This means that the driver is locked in an error state and a hard reset is needed.

Success	Returned when successful.
Failure	Returned when there is a failure from CAAM. An example of this could be a DECO or CCB error.
NoActivityReady	Returned if the CAAM is idle but was expecting it not to be.
Waiting	Returned if waiting on a resource to become available. An example would be if waiting for more entropy.
IllegalRegisterNumber	Error with read register.
OperationNotAllowedOnTheUniversalIODEVice	Writing to registers is not allowed.
TooManyBuffers	The most common case for this return value is either an input/output buffer size is more than expected or more buffers were received than expected.
IllegalStatusNumber	Returned if the argument passed in for AES is not CAAM_ENC or CAAM_DEC.
ArgumentError	Returned if AES key size is not 16, 24, or 32 bytes. Returned if key size is larger than internal buffer to hold it.
SizesTooLarge	Returned if key size, context size, or IV is too large. Also returned in the case that more entropy is requested than available. By default there is 11 word32 registers with entropy giving a max of 44 bytes available at a time.
TransferFailed	Returned in the case that the descriptor being created got larger than 64.

OperationNotImplemented	The type or state attempted to be used is not implemented yet or is unknown.
UsageNotSupported	The type is not supported
ResourceNotAvailable	Returned when there is not an available structure for starting a new descriptor.

3.8 Threaded Applications

There is only a single IORequest available by default which limits the driver to be used by one task at a time. If one task is using the IODevice and another second task from the same process tries to use it at the same time then a ResourceNotAvailable will be returned from the driver to the second task. When wolfSSL receives a ResourceNotAvailable error it will return an error value of WC_HW_WAIT_E.

4.0 Appendix

The appendix has API documentation and benchmark values.

4.1 API

wc_caamSetResource

Synopsis:

wolfssl/wolfcrypt/port/caam/wolfcaam.h

```
int wc_caamSetResource(IODevice ioDev);
```

Description:

This function is used to set the IODevice to make use of hardware acceleration on an i.MX6 board with Integrity OS.

Return Values:

0 is returned on success.

Parameters:

ioDev : This is the Integrity OS IODevice to be used for hardware acceleration.

Example:

```
IODevice ioDev;

/* get IODevice from Integrity OS (default name is wolfSSL_CAAM_Driver) */

ret = wc_caamSetResource(ioDev);
if (ret != 0) {
    // failed to set resource
}
```

See Also:

wolfSSL_Init, wolfCrypt_Init

wc_caamOpenBlob

Synopsis:

wolfssl/wolfcrypt/port/caam/wolfcaam.h

```
int wc_caamOpenBlob(byte* data, word32 dataSz, byte* out, word32* outSz);
```

Description:

This function is used to open a blob. It decrypts the input data using the master key from the hardware and outputs the result. The output buffer must be large enough to hold dataSz - WC_CAAM_BLOB_SZ (macro set to 48). The parameter outSz must be set to the size of the out buffer before passed in.

Return Values:

WC_HW_E : Error case with CAAM driver call.

WC_HW_WAIT_E : Waiting on driver.

BAD_FUNC_ARG : Error case for bad input argument, could be output buffer size or NULL pointers.

0 : Success case.

Parameters:

data : This is the buffer holding the blob to be decoded.

dataSz : Size in bytes of data buffer

out : This is the buffer to hold the result.

outSz : Input as the size in bytes of out buffer. Get set to the exact size of resulting buffer.

Example:

```
byte blob[SIZE];
byte secret[SIZE - WC_CAAM_BLOB_SZ];
word32 secretSz;
int ret;

wolfSSL_Init();

secretSz = sizeof(secret);
ret = wc_caamOpenBlob(blob, sizeof(blob), secret, &secretSz);
if (ret != 0) {
    // failed to open blob
}

// secret now holds the decoded blob
```

See Also:

wc_caamCreateBlob, wolfSSL_Init, wolfCrypt_Init

wc_caamCreateBlob

Synopsis:

wolfssl/wolfcrypt/port/caam/wolfcaam.h

```
int wc_caamCreateBlob(byte* data, word32 dataSz, byte* out, word32* outSz);
```

Description:

This function is used to create a blob. It encrypts the input data using the master key from the hardware and outputs the result. The output buffer must be large enough to hold dataSz + WC_CAAM_BLOB_SZ (macro set to 48). The parameter outSz must be set to the size of the out buffer before passed in.

Return Values:

WC_HW_E : Error case with CAAM driver call.

WC_HW_WAIT_E : Waiting on driver.

BAD_FUNC_ARG : Error case for bad input argument, could be output buffer size or NULL pointers.

0 : Success case.

Parameters:

data : This is the buffer holding the blob to be decoded.

dataSz : Size in bytes of data buffer

out : This is the buffer to hold the result.

outSz : Input as the size in bytes of out buffer. Get set to the exact size of resulting buffer.

Example:

```
byte secret[SIZE];
byte blob[SIZE + WC_CAAM_BLOB_SZ];
word32 blobSz;
int ret;

wolfSSL_Init();

blobSz = sizeof(blob);
ret = wc_caamCreateBlob(secret, sizeof(secret), blob, &blobSz);
if (ret != 0) {
    // failed to create blob
}

// blob now holds the encoded secret
```

See Also:

wc_caamOpenBlob, wolfSSL_Init, wolfCrypt_Init