TEXAS INSTRUMENTS

20450 Century Boulevard
Germantown, MD 20874
Fax: (301) 407-9418

Security Accelerator LLD

Software Design Specification (SDS)

Revision A

QRSA # 00013358

May 24, 2010

Prepared by: Eric Ruei

Revision Record	
Document Title: **Software Design Specification**	
Revision	Description of Change
A	Initial Release
A	- Provide RTSC (IALG-like) system and channel initialization APIs. - Replace SA system context with system API to support multi-instance system - Add system and channel getID APIs - Add some utility APIs - Add action items for known issues
A	- Revised after the unit test - Add new API Sa_getSysStats
A	- Revised with review comments
A	API Updates - Remove IALG dependency - Use C99 types instead of XDC types - Update RNG and PKA related APIs

Note: Be sure the Revision of this document matches the QRSA record Revision letter. The revision letter increments only upon approval via the Quality Record System.

TABLE OF CONTENTS

1 Scope

This document describes the functionality, architecture, and operation of the Security Accelerator Low Level Driver.

2 References

The following references are related to the feature described in this document and shall be consulted as necessary.

No	Referenced Document	Control Number	Description
1	SA LLD PRD 1.0		Product Requirements

Table 1. Referenced Materials

3 Definitions

Acronym	Description
AppSvc	Application Services
API	Application Programming Interface
CPPI	Communication Processor Peripheral Interface
DSP	Digital Signal Processor
LLD	Lower Level Driver
MKI	Master Key Index
MSU	Media Security Unit
PKA	Public Key Accelerator
RNG	Random Number Generator
RTP	Real-time Transport Protocol
QMSS	Queue Manager Sub-System
SA	Security Accelerator
SASS	Security Accelerator Sub-System

Table 2. Definitions

4 Overview

The Security Accelerator (SA) also known as cp_ace (Adaptive Cryptographic Engine) is designed to provide packet security as part of IPSEC, SRTP and 3GPP industry standard. The

SASS low level driver provides an abstraction layer between the application and the Security Accelerator Sub System (SASS). It provides both the system level interface and the channel level interface with a set of APIs defined at section 5.

4.1 Common Interface

The common interface maintains the system level resources and needs to be coordinated among multiple CGEM cores. All the data access provided by the common interface should invoke the SASS CSL layer. The common interface performs the following tasks:

- Reset, download and update the SASS PDSP images.
- Query SASS states and statistics.
- Read a 64-bit true random number
- Perform the large integer arithmetic through the PKA module
- Monitor and report SASS system error

4.2 Channel Interface

The channel interface[1] performs protocol-specific pre-processing for all the packets to be passed to SASS and protocol-specific post-processing for all the packets received from SASS. The channel interface performs the following tasks:

- Convert the channel configuration information into the security contexts defined by the SASS.
- Perform protocol-specific packet operations such as insertion of the ESP header, padding and ESP tail.
- Decrypt and authenticate the received SRTP packet if the SASS is not able to perform the operations due to the key validation failure.
- Generate the command labels in data mode operation.
- Maintain the protocol-specific channel statistics.

The driver does not contain a transport layer and is always non-blocking. The software layers above the SASS LLD must call the appropriate SASS LLD APIs, and then call the appropriate CPPI and QMSS APIs to actually send the data to the SASS.
The interface between the driver, the application, and the SASS is illustrated at Figure 1 .

[1] In TI VoIP DSP Architecture, the channel interface provides MSU-like APIs and functionalities.

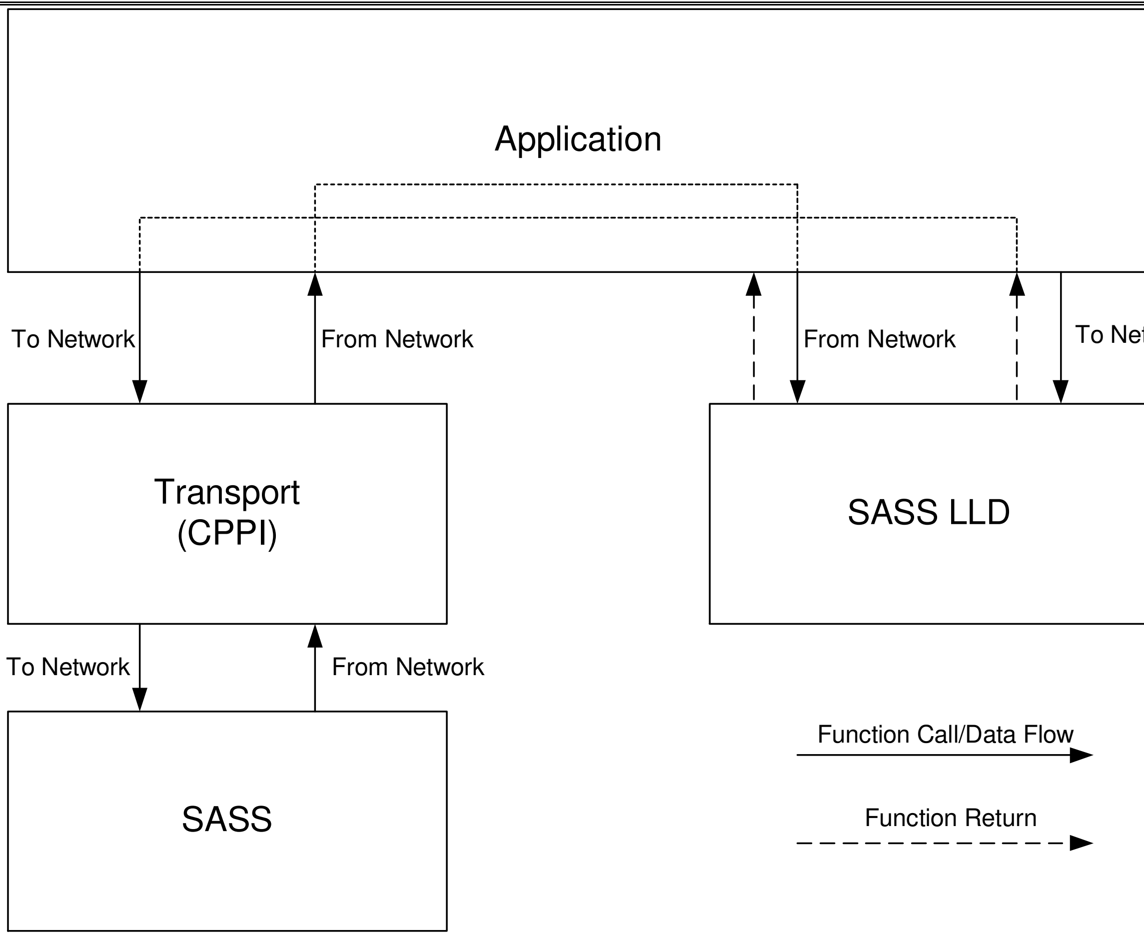

Figure 1. APP/SASS/SASS LLD Data Flow

5 LLD External Interface Definitions (APIs)

5.1 Constant and Type Definitions

5.1.1 SALLD Channel Handle

The SALLD Channel Handle (salldChanHandle_t) is used to identify a SASS channel; it is created and returned by the channel Initialization API.

5.1.2 Protocol Specification

This enumeration defines the security protocol type of the packets to be processed in the SASS.

Name	Description
SALLD_PROT_NULL	No Protocol Specified. The channel will be operating in data mode[2].

[2] In data mode, the user specifies the SASS operations and provides payload information for each packet. There is no protocol-specific packet header parsing performed at SASS.

SALLD_PROT_SRTP	Security RTP
SALLD_PROT_SRTCP	Security RTCP
SALLD_PROT_IPSEC_AH	IPSEC AH mode
SALLD_PROT_IPSEC_ESP	IPSEC ESP mode
SALLD_PROT_3GPP_AC	3GPP Air Ciphering

5.1.3 Channel Size Configuration Information

This structure defines the size configuration information for a security channel.

SALLD_CHAN_SIZE_CONFIG_T:

Name	Description
protocolType	Specify the protocol type of the security channel as defined at section 5.1.2.

5.1.4 Cipher Modes

This enumeration defines the cipher modes used for encryption and decryption. It can be used as an index to the encryption/decryption call table used in software mode, and the encryption and decryption engine software mode control word array used in firmware mode.

Name	Description
SALLD_CIPHER_NULL	No encryption
SALLD_CIPHER_AES_CTR	AES Counter mode
SALLD_CIPHER_AES_F8	AES F8 mode
SALLD_CIPHER_AES_CBC	AES CBC mode
SALLD_CIPHER_DES_CBC	DES CBC mode
SALLD_CIPHER_CCM	Counter with CBC-MAC mode
SALLD_CIPHER_GCM	Galois Counter mode
SALLD_CIPHER_GSM_A53	3GPP GSM A5/3
SALLD_CIPHER_KASUMI_F8	Kasumi F8 mode
SALLD_CIPHER_SNOW_3G_F8	Snow 3G F8 mode

5.1.5 Auth Modes

This enumeration defines the supported authentication modes. It can be used as an index to the authentication call table used in software mode, and the engine software mode control word array used in firmware mode.

Name	Description
SALLD_AUTH_NULL	No Authentication
SALLD_AUTH_MD5	MD5 mode

SALLD_AUTH_SHA1	SHA1 mode
SALLD_AUTH_SHA2_224	224-bit SHA2 mode
SALLD_AUTH_SHA2_256	256-bit SHA2 mode
SALLD_AUTH_HMAC_MD5	HMAC with MD5 mode
SALLD_AUTH_HMAC_SHA1	HMAC with SHA1 mode
SALLD_AUTH_HMAC_SHA2_224	HMAC with 224-bit SHA2 mode
SALLD_AUTH_HMAC_SHA2_256	HMAC with 256-bit SHA2 mode
SALLD_AUTH_GMAC	Galois Counter mode
SALLD_AUTH_CMAC	Cipher-based Message Authentication Code mode
SALLD_AUTH_CBC_MAC	Cipher Block Chaining – Message Authentication Code mode
SALLD_AUTH_KASUMI_F9	Kasumi F9 mode

5.1.6 Channel Configuration Information

This structure defines the configuration information used for security channel creation.

SALLD_CHAN_CONFIG_T:

Name	Description
ID	Specify the logical channel identification
sizeConfig	Specify the size configuration information as defined at section 5.1.3.

5.1.7 General Control Information

The following sub-sections define the channel general control information.

5.1.7.1 Protocol Configuration Information

The following structures define the protocol-specific configuration parameters excluding the keys. There are very few commonly used combinations of key sizes for certain security protocol, such as SRTP and SRTCP. In this case, the user can create table of all the valid combination of keys and use an index to specify the desired key combination.

SALLD_SRTP_CONFIG_PARAMS_T:

Name	Description
masterKeySize	Specify the size of the master key in bytes.
masterSaltSize	Specify the size of the master salt in bytes.
sessionEncKeySize	Specify the size of the session encryption key in bytes.
sessionMacKeySize	Specify the size of the session mac key in bytes.
sessionSaltSize	Specify the size of the session Salt in bytes.
macSize	Specify the size of the authentication tag in bytes.

SALLD_IPSEC_CONFIG_PARAMS_T:

Name	Description
transportType	Specify the transport Type (Transport/Tunnel)
ctrlBitMap	Various control information as specified here: 0x0001: ESN enabled
encryptionBlockSiz	Specify the encryption block size: 1: Stream encryption and no alignment requirement 4: AES-CTR or other stream-like encryption with 4-byte alignment block size: specified by the block encryption algorithm
sessionEncKeySize	Specify the size of the session encryption key in bytes.
sessionMacKeySize	Specify the size of the session mac key in bytes.
sessionSaltSize	Specify the size of the session salt in bytes
ivSize	Specify the size of the initialization vector in bytes.
macSize	Specify the size of the authentication tag in bytes.
nextHdr	Specify the next protocol header
spi	Specify the SPI (Security Parameter Index)
esn	Specify the initial value of the extended sequence Number

SALLD_AC_CONFIG_PARAMS_T:

Name	Description
countC	Specify the high bits, HFN, for the frame Counter: WCDMA RLC AM: the high 20 bits are used WCDMA RLC UM: the high 25 bits are used WCDMA RLC TM: the high 25 bits are used LTE: All 32-bit is used as Count-C GSM: Not used
freash	Specify the 32-bit random number (fresh) required for some integrity check algorithm
ivLow26	Specify the low 26-bit value of the initialization vector
pduType	Specify the Air Ciphering PDU Type (GSM/WCDMA_TMD/WCDMA_UMD/WCDMA_AMD/LTE)
ctrlBitMap	Various control information as specified here: 0x00FF: Bearer Mask 0x0100: Direction (uplink/downlink) = (0/1)
sessionEncKeySize	Specify the size of the session encryption key in bytes
sessionMacKeySize	Specify the size of the session mac key in bytes.
ivSize	Specify the size of the initialization vector in bytes.
macSize	Specify the size of the authentication tag in bytes.

SALLD_DATA_MODE_CONFIG_PARAMS_T:

Name	Description

sessionEncKeySize	Specify the size of the session encryption key in bytes.
sessionMacKeySize	Specify the size of the session mac key in bytes.
sessionSaltSize	Specify the size of the session salt used in GCM/CCM operation in bytes
ivSize	Specify the size of the initialization vector in bytes.
macSize	Specify the size of the authentication tag in bytes.
aadSize	Specify the size of the additional authenticated Data in bytes used in CCM and GCM modes
enc	TRUE: Encryption; FALSE: Decryption
enc1st	TRUE: perform encryption first; FALSE: perform authentication first

SALLD_PROTOCOL_CONFIG_PARAMS_T:

Name	Description
srtp	Specify the configuration parameters for SRTP/SRTCP.
ipsec	Specify the configuration parameters for IPSEC.
ac	Specify the configuration parameters for 3GPP Air Ciphering
data	Specify the configuration parameters in data mode.

5.1.7.2 Destination Information

This structure defines the destination parameters which are used by the SASS to deliver the processed packets to the desired destination.

SALLD_DEST_INFO_T:

Name	Description
flowID	Specify the 8-bit CPPI Flow ID
queueID	Specify the 16-bit destination Queue ID

5.1.7.3 General Configuration Information

This structure defines the general configuration parameters excluding the keys and related re-key parameters.

SALLD_GEN_CONFIG_PARAMS_T:

Name	Description
cipherMode	Specify the cipher mode as defined at section 5.1.4.
authMode	Specify the authentication mode as defined at section 5.1.5.
destInfo	Specify the post-SA destination information
params	Specify the protocol-specific general configuration parameters as defined at section 5.1.7.1.

5.1.7.4 Security General Control Information

This structure defines the general control information of the security channel except of the actual keys which will be configured with a separate data structure, which can be used to reconfigure the security channel with new keys during the call period without affecting the general control information specified here.

SALLD_GEN_CTRL_INFO_T:

Name	Description
validBitfield	The bitmap specifies valid parameters as defined below: • SALLD_CONTROLINFO_VALID_CTRL_BITMAP • SALLD_CONTROLINFO_VALID_TX_CTRL • SALLD_CONTROLINFO_VALID_RX_CTRL • SALLD_CONTROLINFO_VALID_REPLAY_WIN
ctrlBitfield	The bitmap specifies the tx, rx and other operations as defined below: • SALLD_TX_ON: Enable Tx • SALLD_RX_ON: Enable Rx • SALLD_SW_ONLY: Software support only. It is used for the protocols which have not been supported by SASS firmware such as SRTCP.
txCtrl	Specify the general configuration parameters in Tx (To-Network) direction.
rxCtrl	Specify the general configuration parameters in Rx (From-Network) direction.
replayWindowSize	Specify the replay window size

5.1.8 Key Information

The following sub-sections define the Security Key related information. The key related information is part of the LLD channel configuration information. It is defined separately since it will be used frequently during the re-key operation while the general control information stays the same.

5.1.8.1 Protocol Specific Key Information

The following structures define the security key information for each protocol type respectively.

SALLD_SRTP_KEY_PARAMS_T:

Name	Description
ctrlBitfield	Specify the master key types and other control bit definitions as defined below: • SALLD_SRTP_KEY_CTRL_MASTER_KEY

	• SALLD_SRTP_KEY_CTRL_MASTER_SALT • SALLD_SRTP_KEY_CTRL_KEY_DERIVE_RATE • SALLD_SRTP_KEY_CTRL_KEY_LIFETIME • SALLD_SRTP_KEY_CTRL_ROC • SALLD_SRTP_KEY_TYPE_MKI: mki in the packet • SALLD_SRTP_KEY_TYPE_FROM_TO: From-To Key is used
masterKey	Specify the master key.
masterSalt	Specify the master salt.
derivationRate	Specify the key derivation rate in n of 2^n format
keyLifetime	Specify the maximum number of packets allowed for the specified key combination
roc	Specify the initial value of the rollover counter
fromEsn	Specify the starting 48-bit extended sequence number of the specified From-To keys
toEsn	Specify the end 48-bit extended sequence number of the specified From-To keys
mkiSize	Specify the MKI size in bytes
mki	Specify the MKI value

SALLD_IPSEC_KEY_PARAMS_T:

Name	Description
ctrlBitfield	Specify the Key types and other control Information: • SALLD_IPSEC_KEY_CTRL_ENC_KEY • SALLD_IPSEC_KEY_CTRL_MAC_KEY • SALLD_IPSEC_KEY_CTRL_SALT
sesionEncKey	Specify the session encryption key.
sessionAuthKey	Specify the session authentication key
sessionSalt	Specify the session salt

SALLD_AC_KEY_PARAMS_T:

Name	Description
ctrlBitfield	Specify the Key types and other control Information: • SALLD_AC_KEY_CTRL_ENC_KEY • SALLD_AC_KEY_CTRL_MAC_KEY
sesionEncKey	Specify the session encryption key.
sessionAuthKey	Specify the session authentication key

SALLD_DATA_MODE_KEY_PARAMS_T:

Name	Description
ctrlBitfield	Specify the Key types and other control Information: • SALLD_DATA_MODE_KEY_CTRL_ENC_KEY

	• SALLD_DATA_MODE_KEY_CTRL_MAC_KEY • SALLD_DATA_MODE_KEY_CTRL_SALT
sesionEncKey	Specify the session encryption key.
sessionAuthKey	Specify the session authentication key
sessionSalt	Specify the session salt

SALLD_PROTOCOL_KEY_PARAMS_T:

Name	Description
srtp	Specify the key parameters for SRTP/SRTCP
ipsec	Specify the key parameters for IPSEC.
ac	Specify the key related parameters for 3GPP Air Ciphering
data	Specify the key parameters in data mode.

5.1.8.2 Security Key Information

This structure defines the key control information of the security channel. It is used for both initial configuration and re-key operations.

SALLD_KEY_CTRL_INFO_T:

Name	Description
ctrlBitfield	The bitmap Specify the tx, rx and other operations as defined below: • SALLD_TX_KEY_VALID: Tx key is available • SALLD_RX_KEY_VALID: Rx key is available
txKey	Specify the security key parameters in Tx (To-Network) direction.
rxKey	Specify the security key parameters in Rx (From-Network) direction.

5.1.9 Channel Control Information

The following structures define the security channel control related information. They are used by the SASS channel control API as defined at section 5.2.3.

This enumeration defines the channel control types.
SALLD_CHAN_CTRL_TYPE_T:

Name	Description
SALLD_CHAN_CTRL_GEN_CONFIG	SASS channel general configuration
SALLD_CHAN_CTRL_KEY_CONFIG	SASS channel key configuration

This structure defines the channel control information.
SALLD_CHAN_CTRL_INFO_T:

Name	Description

ctrlType	Specify the channel control type as defined above
ctrlInfo	Union of various channel control information including • SALLD_GEN_CTRL_INFO_T • SALLD_KEY_CTRL_INFO_T

5.1.10 SASS Packet Error

This enumeration lists all the packet error conditions which may occur in the SASS.

SALLD_PKT_ERR_T:

Name	Description
SALLD_PKT_ERR_OK	No Error
SALLD_PKT_ERR_REPLAY_OLD	The packet is rejected by SASS because it is out of the replay window range
SALLD_PKT_ERR_REPLY_DUP	The packet is rejected by SASS because it is duplicated.
SALLD_PKT_ERR_INVALID_KEY	The packet is rejected by SASS because the key in the security context is out of range
SALLD_PKT_ERR_INVALID_MKI	The packet is rejected by SASS because the MKI mismatches
SALLD_PKT_ERR_AUTH_FAIL	The packet is rejected by SASS because the Authentication (hash) verification fails.

5.1.11 Packet Descriptor Information

It is important to pass the packet header parsing information among the PASS, SASS, SA LLD and application to avoid re-parsing the packet multiple times. The following structures define the packet descriptor information. It may be defined in a global common header file to be shared among modules.

SA_PKT_DESC_T:

Name	Description
size	Specify the packet length.
payloadOffset	Specify the offset from the top of the packet to the protocol header. IPSEC ESP/AH: IP header SRTP: RTP header SRTCP: RTCP header 3GPP Air Ciphering: PDU header
payloadLen	Specify the total payload length to be processed
nSegments	Specify number of data segments in the packet
segments	Pointer to the array which contain the segment addresses
segUsedSizes	Pointer to the array which contain the used size in each data

	segment
segAlloSizes	Pointer to the array which contain the allocated size in each data segment

5.1.12 SALLD SW Information

This structure contains the SA-specific software information required for all packets to be delivered to SA. It will be provided by the SA LLD based on the general channel configuration parameters and the security protocols. The software information words should be copied to the CPPI Software words area by the CPPI LLD or equivalent as provided.

SALLD_SW_INFO_T:

Name	Description
size	Specify the software info size in 32-bit words
swInfo	Contain the software info words required by SA

5.1.13 Command label Information

This structure defines the SA-specific command label information used by SASS to route packets to sub-engines within SASS. The command label information will be formatted by the SA LLD based on the channel configuration parameters and the payload information as defined at section 5.1.14 in data mode. The command label should be copied to the protocol-specific section at the CPPI packet descriptor as provided through the CPPI LLD or equivalent component.

SALLD_CMD_LABEL_INFO_T:

Name	Description
cmdLbSize	Specify the size of the command labels in bytes.
cmdLbBuf	Pointer to the buffer which contains the command labels

5.1.14 Payload Information

This structure defines the payload related information used to construct the SA-specific command labels in data mode.

SALLD_PAYLOAD_INFO_T:

Name	Description
encOffset	Specify the offset to the encrypted/decrypted data in the packet in bytes.
authOffset	Specify the offset to the authenticated data in the packet in bytes
encSize	Specify the total number of bytes to be encrypted or decrypted
authSize	Specify the total number of bytes to be authenticated
encIV	Contain the initialization vectors used in certain encryption mode. Note: IV should be specified here in GCM/CCM mode

authIV	Contain the initialization vectors used in certain authentication mode. Note: IV should be specified here in GMAC mode
aad	Contain the additional authenticated data in GCM/CCM modes

5.1.15 Packet Information

This structure defines the input and output packet formats. It may be converted to more generic packet format specified by the network driver or equivalent.

SALLD_PKT_INFO_T:

Name	Description
pktDesc	Specify packet descriptor
validBitmap	Specify which parameter is valid as defined below: • SALLD_PKT_INFO_VALID_RX_ERR_CODE • SALLD_PKT_INFO_VALID_SW_INFO • SALLD_PKT_INFO_VALID_CMDLB_INFO • SALLD_PKT_INFO_VALID_PAYLOAD_INFO
pktErrCode	Specify the received error code from SASS. It is only valid in the from-network direction
swInfo	Store the software information required by SA as defined at section 5.1.12. It is only valid in to-network direction.
cmdLbInfo	Command label information as defined at section 5.1.13. It is only valid in to-network direction.
payloadInfo	Specify the payload information in data mode

5.1.16 Key Request Information

The following structures define the Security Key request information. It is only used for SRTP re-key operation at this moment. However, protocol specific structures are defined so that it can be used for other protocols if required.

SALLD_SRTP_KEY_REQUEST_T:

Name	Description

ctrlBitfield	The bitmap Specify the master key types and other control bit definitions as defined below:
	• SALLD_SRTP_KEY_TYPE_MKI:
	    o Set: MKI key is used
	    o Clear: From-To key is used
	• SALLD_SRTP_KEY_MKI_VALID: MKI Value is specified
	• SALLD_SRTP_KEY_TX: Tx key
	• SALLD_SRTP_KEY_RX: Rx key

mki	Specify the mki value of the MKI keys

SALLD_KEY_REQUEST_T:

Name	Description
srtp	Specify the key request parameters for SRTP.

5.1.17 Security Context Information

The 16-bit security context ID and its corresponding context buffer which contains the security context parameters should be maintained across multiple DSP cores. Therefore, they are multi-core system resources and should be managed by the application. This structure is used to exchange security context information between SA LLD and the application.

SALLD_SC_INFO_T:

Name	Description
scID	Security Context ID specified by the application
scBuf	Security Context Buffer provided by the application. It is required to be 16-byte aligned.
scSize	Specify the required size of the security context

5.1.18 SASS Reset State and Query Argument

This enumeration defines SA Sub-System queries and states (see section 5.3.4)
SALLD_STATE_RESET
SALLD_STATE_ENABLE
SALLD_STATE_QUERY
SALLD_STATE_INCONSISTENT
SALLD_STATE_INVALID_REQUEST
SALLD_STATE_ENABLE_FAILED

5.1.19 PKA Request Information

The following structures define the PKA request information which is used to perform the large vector arithmetic operations.

This enumeration defines the PKA arithmetic operation.
SALLD_PKA_OP_T:

Name
SALLD_PKA_OP_ADD
SALLD_PKA_OP_SUB
SALLD_PKA_OP_MUL
SALLD_PKA_OP_DIV
SALLD_PKA_OP_RSHIFT
SALLD_PKA_OP_LSHIFT
SALLD_PKA_OP_COMPARE

SALLD_PKA_OP_COPY
SALLD_PKA_OP_EXP2
SALLD_PKA_OP_EXP4

This enumeration defines the PKA comparison result.

SALLD_PKA_CMP_T

Name	Description
SALLD_PKA_CMP_AEQB	A = B
SALLD_PKA_CMP_ASUPB	A > B
SALLD_PKA_CMP_AINFB	A < B

SALLD_PKA_REQ_INFO_T

Name	Description
operation	Specify the arithmetic operation as defined above
oprandPtr[3]	Pointers to up to 3 operands. Note: Number of valid operands is based on the operation
oprandSize[3]	Specify the size of operand in 32-bit words
remSize	Specify the size of reminder in 32-bit words
resultSize	Specify the size of reminder in 32-bit words
remPtr	Pointer to the reminder
resultPtr	Pointer to the arithmetic result
numShiftBits	Number of bits in shift operation
cmpResult	Result of comparison

5.1.20 RNG Configuration Information

The following structures define the RNG configuration information.

SALLD_RNG_CONFIG_PARAMS_T:

Name	Description

ctrlBitfield	The bitmap specify the initialization mode and other control bit definitions as defined below:
	• SALLD_RNG_CTRL_REINIT:
	  o Set: Force re-initialization
	  o Clear: Initialize RNG only if it has not been initialized
	• SALLD_RNG_CTRL_ENABLE_INT:
	  o Set: Interrupt Mode
	  o Clear: Poll Mode
	• SALLD_RNG_CTRL_RESET: Software reset

clockDiv	Specify the clock divider (1, 16) 0: default
startupCycles	Specify the number of clock cycles (2**8, 2**24) to start up the random number generator initially(0:default)
minRefillCycles	Specify the minimum number of clock cycles (2**6, 2**14) to generate the next random number (0:default)
maxRefillCycles	Specify the maximum number of clock cycles (2**8, 2**24) to generate the next random number (0:default)

5.1.21 System Size Configuration Information

This structure defines the size configuration information for a SA instance.

SALLD_SIZE_CONFIG_T:

Name	Description
nMaxChan	Specify the maximum number of channels supported

5.1.22 System Configuration Information

This structure defines the configuration information when a SA instance is created.

SALLD_CONFIG_T:

Name	Description
ID	Specify the logical instance identification
callTable	Pointer to call-out function Table as defined at section 5.4

5.2 Call-in APIs

APIs are called by application or higher level drivers (typically configuration APIs and transmit to peripheral type APIs). The APIs listed in this section are presented as pseudo code. In cases where functions have large argument lists it is likely that one or more structures will be used instead of the argument list.

The SA LLD channel interface provides the call-in APIs described from section 5.2.1 to section 5.2.8. The SA LLD common interface provides the call-in APIs described from section 5.2.9 to section 5.2.21. The SA LLD also provides some utility APIs described from section 5.2.22.

5.2.1 SASS Channel Get Buffer Description

5.2.1.1 Purpose/Description

The function is called by the application to inquire the specific memory requirement of the SASS channel.

5.2.1.2 Prototype

```
Result = Sa_chanGetBufferReq(
                *sizeCfg,       /* channel size configuration information defined at section 5.1.3*/
                size[],         /* array of size requirements */
               align[]);        /* array of alignment requirements */
        );
```

5.2.1.3 Implementation

The memory size and alignment requirements are calculated and returned. The application must allocate the size and alignment requirement array with the number of elements defined by sa_CHAN_N_BUFS. The application must allocate the requested memory and provide it back to the driver in the call to Sa_chanCreate.

5.2.1.4 Return Values

SALLD_ERR_OK
SALLD_ERR_PARAMS.

5.2.2 SASS Channel Creation

5.2.2.1 Purpose/Description

The function creates a SASS channel and initializes the corresponding instance structure based on the channel configuration data including the SASS protocol type and etc. Other configuration information will be provided through the SASS channel control API when the information is available at the application.

5.2.2.2 Prototype

```
Result = Sa_chanCreate(
                handle ,          /* Handle of its parent SALLD instance */
                *chanCfgInfo , /* channel configuration information defined at section 5.1.6 */
                *bases,           /* Array of the memory buffer base addresses */
                *pChanHdl        /* Store the channel handle */
        );
```

5.2.2.3 Implementation

The chanCreateAPI is the combination of the tranditional chanNew and chanOpen APIs. It performs the following actions:

- Verify the allocated memory buffers match the SASS channel memory requirements
- Initialize the common channel instance
- Initialize the instance call table based on the channel protocol type
- Initialize the protocol-specific channel instance

5.2.2.4 Return Values

SALLD_ERR_OK
SALLD_ERR_INVALID_BUF
SALLD_ERR_INVALID_PROTO_TYPE

5.2.3 SASS Channel Control

5.2.3.1 Purpose/Description

The function is called to configure or re-configure the SASS channel with various control information. Only the control type SALLD_CHAN_CTRL_GEN_CONFIG and SALLD_CHAN_CTRL_KEY_CONFIG are supported. In the typical call flow, the function will be called with both control types multiple times to configure and activate the SASS channel during the call setup period and then it should be called with SALLD_CHAN_CTRL_KEY_CONFIG only to perform re-key operation.

5.2.3.2 Prototype

```
Result = Sa_chanContro(
                sassHandle,     /* channel identifier */
                chanCtrlInfo    /* channel control information  defined at section  5.1.9 */
          );
```

5.2.3.3 Implementation

The SASS Channel Control API behaves differently based on the protocol type. The protocol-specific implementations will be accessed through the call table. They perform the following actions:

- Enable/Disable Tx/Rx channel.
- Convert and record the configuration information at the channel instance

- Activate the channel when all the configuration information is available
  - Call application to request the security context ID and buffer and record them at the channel instance
  - Convert the configuration information into protocol-specific SASS security context
  - Call application to register the security context ID and buffer pointer for Rx (from-Network) operation
  - Set channel state to ACTIVE
- Allocate buffer and record key information during the re-key operation

5.2.3.4 Return Values

SALLD_ERR_OK
SALLD_ERR_OUT_OF_CTX_BUF

5.2.4 SASS Channel Send Data

5.2.4.1 Purpose/Description

The function processes the data packet in the To-Network direction. It performs protocol-specific operations to prepare the data packet to be encrypted, decrypted and/or integrity protected or checked by the SASS. It also performs the actual encryption and/or integrity protection for RTP/RTCP packets in the SW[3] only mode.

5.2.4.2 Prototype

```
Result = Sa_chanSendData(
            sassHandle,    /* channel identifier */
            pktInfo ,      /* packet information defined at section 5.1.14 */
            clear          /* TRUE: force non-encryption */
        );
```

5.2.4.3 Implementation

The protocol specific implementations described at the sub-sections will be accessed through the call table.

5.2.4.3.1 SRTP

It performs the following actions in Firmware mode

- Perform Re-key operation
  - Verify whether the master key is expired.
    - If the masker key is expired and the new key is not available, call API to request new key and return error.
    - If the masker key is expired and the new key is available, derive the new session keys and generate the new SASS security context.

[3] The SW only mode is only applicable to SRTP and SRTCP.

    - If the session key is expired, derive the new session keys and generate the new SASS security context.
- Generate SRTP padding if necessary
- Update the packet size and protocol (TCP/UDP) payload size in the packet descriptor to reserve room for the MKI and authentication tag
- Prepare the CPPI Software information as defined at section 5.1.12.
- Update statistics

It performs the following actions in SW only mode

- Perform Re-key operation
    - Verify whether the master key is expired.
        - If the masker key is expired and the new key is not available, call API to request new key and return error.
        - If the masker key is expired and the new key is available, derive the new session keys.
    - If the session key is expired, derive the new session keys.
- Generate SRTP padding if necessary
- Perform data encryption based on the specified cipher mode.
- Append the roc at the end of the packet and perform authentication operation based on the specified mac mode.
- Append the MKI and the authentication tag at the end of packet.
- Update the packet size and protocol (TCP/UDP) payload size in the packet descriptor
- Update statistics

5.2.4.3.2 **IPSEC AH**

It performs the following actions:

- Reserve room for the Authentication Header (the next header, AH length, SPI, sequence number, IV and the authentication data) following the external IP header for Tunnel mode or the original IP header for transport mode.
- Perform ICV padding for IPV6 if necessary since the AH header should be multiple of 8-bytes
- Extract the next header (protocol) from the IP header, and replace it with AH Transport (51)
- Fill in the next header and the AH length.
- Adjust the payload length of the external IP header.
- Update the packet size and protocol (IP) payload size in the packet descriptor to reserve room for the ESN and payload padding if necessary
- Prepare the CPPI Software information as defined at section 5.1.12.

5.2.4.3.3 **IPSEC ESP**

It performs the following actions:

- Reserve room for the ESP Header (SPI and sequence number) and IV following the external IP header for Tunnel mode or the original IP header for transport mode.

- Extract the protocol (next header) from the original IP header, and replace it with ESP Transport (50).
- Calculate the ESP padding size, insert ESP padding and the ESP Trail (padding size + next header)
- Adjust the payload length of the external IP header
- Update the packet size and protocol (IP) payload size in the packet descriptor to reserve room for the ESN and authentication data if necessary
- Prepare the CPPI Software information as defined at section 5.1.12.

5.2.4.3.4 3GPP Air Ciphering

It performs the following actions:

- Prepare the CPPI Software information as defined at section 5.1.12.
- Update statistics

5.2.4.3.5 Data

It performs the following actions:

- Prepare the CPPI Software information as defined at section 5.1.12.
- Prepare the Command Label information as defined at section 5.1.13.
- Update statistics

5.2.4.4 Return Values

SALLD_ERR_OK
SALLD_ERR_OUT_OF_CTX_BUF
SALLD_ERR_SRTP_NO_KEY

5.2.5 SASS Channel Receive Data

5.2.5.1 Purpose/Description

The function processes the data packet in the From-Network direction. It performs protocol-specific post-SASS operations on the decrypted/encrypted and /or integrity-checked data packet. It also performs the actual decryption and/or integrity check for the SRTP/SRTCP packets in the SW[4] only mode.

5.2.5.2 Prototype

```
Result = Sa_chanReceiveData(
                sassHandle,     /* channel identifier */
                pktInfo         /* packet information defined at section 5.1.14 */
        );
```

[4] The SW only mode is only applicable to SRTP and SRTCP.

5.2.5.3 Implementation

The protocol specific implementations described at the sub-sections will be accessed through the call table.

5.2.5.3.1 SRTP

To provide fully-offloaded SRTP operation in the From-Network direction, the SASS is equipped to perform key validation and pass the original packet to the CGEM when the key validation fails. This function will perform the general re-key operation and the authentication and decryption operation if new keys are available when error packet is received.

It performs the following actions in Firmware mode

- Process the packet with SASS error code SALLD_PKT_ERR_INVALID_KEY
    - Verify whether the master key is expired.
        - If the masker key is expired and the new key is not available, call API to request new key and return error.
        - If the masker key is expired and the new key is available, derive the new session keys and generate the new SASS security context.
    - If the session key is expired, derive the new session keys and generate the new SASS security context.
    - Record the authentication tag and remove MKI and authentication tag from the packet.
    - Append the roc at the end of packet and perform authentication operation based on the specified auth mode and the new authentication session key.
    - Perform data decryption based on the specified cipher mode and the new session keys if the authentication tag matches. Otherwise, update the statistics and return error.
    - Change internal state to the key transition state. Stay in this state until the replay window base is within the new range. Call API to register the new security context and enter normal state.
- Process the packet with SASS error code SALLD_PKT_ERR_INVALID_MKI
    - If the new key is not available, call API to request new key and return error.
    - If the new key is available and the MKI matches, derive the new session keys and generate the new SASS security context.
    - Record the authentication tag and remove MKI and authentication tag from the packet.
    - Append the roc at the end of packet and perform authentication operation based on the specified auth mode and the new authentication session keys.
    - Perform data decryption based on the specified cipher mode and the new session keys if the authentication tag matches. Otherwise, update the statistics and return error.
    - Change internal state to the key transition state. Stay in this state until the replay window base is within the new range. Call API to register the new security context and enter normal state.

- Process the packet with SASS error code SALLD_PKT_ERR_REPLAY_OLD, SALLD_PKT_ERR_REPLAY_DUP or SALLD_PKT_ERR_AUTH_FAIL
    - Update statistics and return Error
- Process the packet without Error SALLD_PKT_ERR_OK
    - Remove the MKI and authentication tag
- Replay window updates
- Update the packet size and protocol (TCP/UDP) payload size in the packet descriptor
- Update statistics

It performs the following actions in SW only mode

- Perform Re-key operation
    - Verify whether the master key is expired.
        - If the masker key is expired and the new key is not available, call API to request new key and return error.
        - If the masker key is expired and the new key is available, derive the new session keys.
    - If the session key is expired, derive the new session keys.
- Record the authentication tag and remove MKI and authentication tag from the packet.
- Append the roc at the end of the packet and perform authentication operation based on the specified auth mode.
- Perform data decryption based on the specified cipher mode if the authentication tag matches. Otherwise, update the statistics and return error.
- Remove the MKI and the authentication tag at the end of packet.
- Update the packet size and protocol (TCP/UDP) payload size in the packet descriptor.
- Replay window updates
- Update statistics

5.2.5.3.2 **IPSEC AH**

It performs the following actions:

- Return Error if the SASS packet error occurs.
- Extract the next header from the AH header and replace the one in the IP header with it.
- Update the packet size and protocol (IP) payload size in the packet descriptor by removing the size of the AH header
- Remove the AH Header

5.2.5.3.3 **IPSEC ESP**

It performs the following actions:

- Return Error if the SASS packet error occurs.
- Verify the ESP padding bytes
- Extract the next header from the ESP Trailer and replace the one in the IP header with it.

- Update the packet size and protocol (IP) payload size in the packet descriptor by removing the size of the authentication data, the ESP header, padding and the ESP trailer.
- Remove the ESP header, Trailer and the Authentication data
- Update statistics

5.2.5.3.4 3GPP Air Ciphering

It performs the following actions:

- Update the corresponding statistics and return Error if the SASS packet error occurs.
- Update statistics

5.2.5.3.5 Data

This function should not be used in data mode.

5.2.5.4 Return Values

SALLD_ERR_OK
SALLD_ERR_OUT_OF_CTX_BUF
SALLD_ERR_SRTP_NO_KEY
SALLD_ERR_REPLAY_OLD
SALLD_ERR_REPLAY_DUP
SALLD_ERR_AUTH_FAIL
SALLD_ERR_PADDING_FAIL

5.2.6 SASS Channel Statistics Request

5.2.6.1 Purpose/Description

The function queries the SASS channel statistics. Some statistics can be clear if the reset flag is set.

5.2.6.2 Prototype

```
Result = Sa_chanGetStats(
            sassHandle,      /* channel identifier */
            flags,           /* STATS_QUERY_FLAG_CLEAR: Clear some statistics
                             after they are reported
                             STATS_QUERY_FLAG_TRIG: Inform the SASS to
                             provide statistics
                             STATS_QUERY_FLAG_NOW: Request the SA LLD to
                             provide the latest available statistic */
            *stats           /*  pointer to the buffer where the protocol specific
      statistics
                              *  to be copied to */
   );
```

5.2.6.3 Implementation

Perform the following actions:

- Invoke send null-packet API to trigger SASS to report the channel statistics if STATS_QUERY_FLAG_TRIG is set
- Copy the requested common statistics if available
- Copy the requested protocol-specific statistics if available
- Reset the internal statistics which can be clear when the clear flag is set

5.2.6.4 Return Values

SALLD_ERR_OK
SALLD_ERR_STATS_UNAVAIL

5.2.7 SASS Channel Close

5.2.7.1 Purpose/Description

The function deletes a SASS channel and releases all the internal resources and returns the memory buffer information so that the application may free the allocated buffers.

5.2.7.2 Prototype

```
Result = Sa_chanClose(sassHandle , /* channel identifier */
                      *bases);      /* output array of the memory buffer base addresses*/
```

5.2.7.3 Implementation

The chanFree API is the combination of the tranditional chanClose and chanDelete APIs. It performs the following actions:

- Clear and release the protocol-specific resources and buffers.
- Return the memory buffer requests of the SASS channel so that they can be freed by the application.

5.2.7.4 Return Values

SALLD_ERR_OK

5.2.8 SASS Get Channel ID

5.2.8.1 Purpose/Description

The function returns the SA channel ID associated with the channel handle.

5.2.8.2 Prototype

```
Result = Sa_chanGetID (sassHandle  ) /* channel identifier */
```

5.2.8.3 Implementation

Extract and return the channel ID from the channel instance.

5.2.8.4 Return Values

SA channel ID

5.2.9 SASS Get Buffer Description

5.2.9.1 Purpose/Description

The function is called by the application to inquire the specific memory requirement of the SASS instance.

5.2.9.2 Prototype

```
Result = Sa_getBufferReq(
        *sizeCfg,      /* channel size configuration information defined at section 5.1.20
                       */
         size[],       /* array of size requirements */
        align[]);      /* array of alignment requirements */
    );
```

5.2.9.3 Implementation

The memory size and alignment requirements are calculated and returned. The application must allocate the size and alignment requirement array with the number of elements defined by sa_N_BUFS. The application must allocate the requested memory and provide it back to the driver in the call to Sa_create.

5.2.9.4 Return Values

SALLD_ERR_OK
SALLD_ERR_PARAMS

5.2.10 SASS Creation

5.2.10.1 Purpose/Description

The function creates a SASS instance and initializes the corresponding instance structure based on the system configuration data including the SASS call-out function table and etc.

5.2.10.2 Prototype

```
Result = Sa_create(
        *cfgInfo       /* channel configuration information defined at section 5.1.22 */
        *bases,        /* Array of the memory buffer base addresses */
        *pHandle       /* Store the  system handle */
    );
```

5.2.10.3 Implementation

The system Init API is the combination of the traditional system New and Open APIs. It performs the following actions:

- Verify the allocated memory buffers match the SASS system memory requirements
- Initialize the system instance
- Verify and initialize the call-out function table

5.2.10.4 Return Values

SALLD_ERR_OK
SALLD_ERR_PARAMS
SALLD_ERR_NOMEM
SALLD_ERR_INVALID_BUF

5.2.11 SASS Close

5.2.11.1 Purpose/Description

The function deletes a SASS instance and releases all the internal resources and returns the memory buffer information so that the application may free the allocated buffers.

5.2.11.2 Prototype

```
Result = Sa_close(handle ,     /*  instance identifier */
                  *memTab);    /* Output array of memory buffer base addresses */
```

5.2.11.3 Implementation

The system Close API is the combination of the traditional system Close and Delete APIs. It performs the following actions:

- Clear and release the instance resources and buffers.
- Return the memory buffer requests of the SASS instance so that they may be freed by the application.

5.2.11.4 Return Values

SALLD_ERR_OK

5.2.12 SASS System Statistics Request

5.2.12.1 Purpose/Description

The function queries the SASS systeml statistics.

5.2.12.2 Prototype

```
Result = Sa_getSysStats(
            handle,      /* SALLD instance identifier */
            *stats       /*  pointer to the buffer where the system statistics
```

```
                        *  to be copied to */
        );
```

5.2.12.3 Implementation

Perform the following actions:

- Extract and copy the system statistics from the SASS

5.2.12.4 Return Values

SALLD_ERR_OK

5.2.13 Reset/Enable/Get SASS state

5.2.13.1 Description

This function will set or release reset for the SASS subsystem. Because this command results in direct access to the SASS it could be blocking. Because of this a macro is provided which the system can use to provide its own transport mechanism.

5.2.13.2 Prototype

```
Result = Sa_resetControl (
             handle,      /* SALLD instance identifier */
             newState   /* specify the new state as defined below */
             );
```

The following values are used both for the input argument and the result.

```
Enum newState  {
  SALLD_STATE_RESET,                  /* Input arg and output result */
  SALLD_STATE_ENABLE,                 /* Input arg and output result */
  SALLD_STATE_QUERY,                  /* Input arg only              */
  SALLD_STATE_INCONSISTENT,           /* Output only                 */
  SALLD_STATE_INVALID_REQUEST,        /* Output only                 */
  SALLD_STATE_ENABLE_FAILED           /* Output only                 */
};
```

5.2.13.3 Implementation

The function uses the CSL layer to make direct writes to the SASS control registers. Because these registers exist in config space and a read is always required, this command can block while the config bus is unavailable.

5.2.13.4 Return value

The return values are the same as the input value, except that SALLD_STATE_QUERY will never be returned. The value SALLD_STATE_INCONSISENT means that some of the elements of the SASS are out of reset, but others are in reset.

5.2.14 Download SASS PDSP Image

5.2.14.1 Description

This function is used to download a PDSP image to one of the two PDSPs. Because this function call results in direct access to the SASS it could be blocking. Because of this a macro is provided which the system can use to provide its own transport mechanism.

5.2.14.2 Prototype

```
Result = Sa_downloadImage (
            handle,      /* SALLD instance identifier */
            moduleId,    /* Identifies the PDSP */
            address,     /* The image source */
            sizeBytes    /* The image size */
        );
```

5.2.14.3 Implementation

The LLD will check the state of the PDSP prior to writing the image. An error is returned if the specified PDSP is not in reset. Otherwise the image is copied to the PDSP program memory using direct writes.

5.2.14.4 Return values

SALLD_ERR_OK
SALLD_ERR_SYS_INVALID_STATE

5.2.15 SASS Init RNG

5.2.15.1 Purpose/Description

The function is called to configure and initialize the RNG (Random Number Generator) module inside SASS. For a multi-core device, it is up to the upper-layer application to make sure that only the master core performs the RNG hardware initialization.

5.2.15.2 Prototype

```
Result = Sa_rngInit(
            handle,     /* SALLD instance identifier */
            *cfg        /* Pointer to the configuration structure defined at 5.1.20 */
        );
```

5.2.15.3 Implementation

Initialize the system-level variables and invoke CSL registers to reset and configure the RNG module if hardware initialization is required.

5.2.15.4 Return Values

SALLD_ERR_OK

SALLD_ERR_PARAMS

5.2.16 SASS Get Random Number

5.2.16.1 Purpose/Description

The function is used to get a 64-bit true random number. It is a non-blocking function call with return code which indicates whether the random number is available.

5.2.16.2 Prototype

```
Result = Sa_getRandonNum(
                handle,      /* SALLD instance identifier */
               *number       /* function return: 64-bit random number */
           );
```

5.2.16.3 Implementation

It performs the following actions:

- Call the CSL function to check the RNG status
- If the RNG is busy, return the error code to indicate that RNG is unavailable
- If the RNG is ready, call the CSL function to retrieve the random number.

5.2.16.4 Return Values

SALLD_ERR_OK
SALLD_ERR_PARAMS
SALLD_ERR_MODULE_BUSY
SALLD_ERR_MODULE_UNAVAIL

5.2.17 SASS Close RNG

5.2.17.1 Purpose/Description

The function is called to deactivate the RNG module and clears its corresponding internal state. For a multi-core device, it is up to the upper-layer application to make sure that only the master core performs the RNG hardware shutdown.

5.2.17.2 Prototype

```
Result = Sa_rngClose(
                handle       /* SALLD instance identifier */
            );
```

5.2.17.3 Implementation

Reset the system-level variables and invoke CSL registers to disable the RNG module..

5.2.17.4 Return Values

SALLD_ERR_OK

5.2.18 SASS Init PKA

5.2.18.1 Purpose/Description

The function is called to initialize the PKA module inside SA. For a multi-core device, it is up to the upper-layer application to make sure that only the master core performs the PKA hardware initialization.

5.2.18.2 Prototype

```
Result = Sa_pkaInit(
                Handle        /* SALLD instance identifier */
           );
```

5.2.18.3 Implementation

Initialize the system-level variables and invoke CSL functions to reset the PKA if hardware initialization is required.

5.2.18.4 Return Values

SALLD_ERR_OK

5.2.19 SASS PKA Operation

5.2.19.1 Purpose/Description

The function is called to perform a large vector arithmetic operation through the PKA module. It is considered as a blocking function call since it will wait for the PKA module to complete the arithmetic operation. However, it only takes a few cycles for PKA to complete any operation. This function also returns with error code immediately if the PKA is still in the process to perform the previous operation. For multi-core device, it is up to the application to prevent this function to be invoked by multiple CGEM cores simultaneously.

5.2.19.2 Prototype

```
Result = Sa_pkaOperation(
                handle,       /* SALLD instance identifier */
                pkaReqInfo  /* PKA Request Information as defined at section 5.1.19 */
           );
```

5.2.19.3 Implementation

It performs the following actions:

- Call the CSL function to check the PKA status
- If the PKA is busy, return the error code to indicate that PKA is unavailable
- If the PKA is ready, call the CSL functions to access the PKA registers and vector RAM to prepare and trigger the specified operation. Copy the operation result from the PKA registers and vector RAM to the PKA request information data structure.

5.2.19.4 Return Values

SALLD_ERR_OK
SALLD_ERR_PKA_UNAVAIL

5.2.20 SASS Close PKA

5.2.20.1 Purpose/Description

The function is called to deactivate the PKA module and clears its corresponding internal state. For a multi-core device, it is up to the upper-layer application to make sure that only the master core performs the PKA hardware shutdown.

5.2.20.2 Prototype

```
Result = Sa_pkaClose(
                handle      /* SALLD instance identifier */
          );
```

5.2.20.3 Implementation

Reset the system-level variables and invoke CSL registers to disable the PKA module..

5.2.20.4 Return Values

SALLD_ERR_OK

5.2.21 SASS Get System ID

5.2.21.1 Purpose/Description

The function returns the SA instance ID associated with the instance handle.

5.2.21.2 Prototype

```
Result = Sa_getID (handle) /* SALLD instance identifier */
```

5.2.21.3 Implementation

Extract and return the system ID from the instance.

5.2.21.4 Return Values

SA system ID

5.2.22 SASS Verify Security Context Buffer

5.2.22.1 Purpose/Description

The function verifies whether the security context buffer is freed by the SASS.

5.2.22.2 Prototype

Result = Sa_isScBufFree (*scBuf) /* Pointer to the Security Context buffer */

5.2.22.3 Implementation

Verify the owner bit of the control bitmap at the security context buffer.

5.2.22.4 Return Values

TRUE if buffer is free; FALSE if otherwise

5.3 Macros

To separate the commands from the transport layer, each macro makes use of the following macros that must be defined outside the system. These macros are used by the sass utility macros in this section.

SYSTEM_WRITE32(address32, uvalue32)
SYSTEM_COPY(destAddr32, srcAddr32, sizeBytes)
SYSTEM_READ32(address32)

5.3.1 Reset Subsystem

5.3.1.1 Description

The SASS consists of multiple PDSPs and hardware sub-modules. A subsystem reset will assert reset all of these modules.

5.3.1.2 Invocation

SALLD_RESET_SUBSYSTEM()

5.3.1.3 Implementation

The macro performs multiple invocations of SYSTEM_READ32 and SYSTEM_WRITE32 to put every PDSP into reset.

5.3.2 Enable Subsystem

5.3.2.1 Description

Enable subsystem takes all systems out of reset

5.3.2.2 Invocation

SALLD_ENABLE_SUBSYSTEM()

5.3.2.3 Implementation

The macro performs multiple invocations of SYSTEM_READ32 and SYSTEM_WRITE32 to release every PDSP from reset.

5.3.3 Download Image

5.3.3.1 Description

An image is downloaded into a single PDSP. The PDSP must be in reset or unpredictable behavior from the SASS could occur.

5.3.3.2 Invocation

SALLD_DOWNLOAD_MODULE(moduleId, address32, sizeBytes)

5.3.3.3 Implementation

The macro invokes SYSTEM_COPY to copy the image to the desired module. Module IDs have values 0-1 but are not symbolically enumerated.

5.3.4 Get Reset State

5.3.4.1 Description

This macro returns the reset state of the SASS.

5.3.4.2 Invocation

SALLD_GET_SYSTEM_STATE()

5.3.4.3 Implementation

The macro invokes the SYSTEM_READ32 macro to determine the operational state of the subsystem. The result is a value as defined in section 5.1.18.

5.4 Call-Out APIs

The un-restricted call-out APIs may cause OS context switch or unexpected long call chain which requires deep stack. To avoid those problems, most of the call-out APIs defined below can be invoked by the application at the end of some call-in APIs based on its return value and/or control flags. Other call-out APIs should be implemented as simple non-blocking functions.

5.4.1 SASS Channel Key Request

5.4.1.1 Purpose/Description

The function is invoked to request a new security key. It may be triggered by either the Sa_chanSendData() or Sa_chanReceiveData() API. The application should call the SASS Channel Control function to pass the new key when it is available.

5.4.1.2 Prototype

```
*ChanKeyRequest(sassHandle,          /* channel identifier */
                chanKeyRequest       /* key request information as defined at section 5.1.16*/
                );
```

5.4.2 SASS Security Context Allocation

5.4.2.1 Purpose/Description

The function is called to request the application to allocate the security context with the specified size. It must be implemented as a simple non-blocking function.

5.4.2.2 Prototype

```
*ScAlloc(sassHandle   /* channel identifier */
         chanScInfo   /* security context request information as defined at section 5.1.17 */
         );
```

5.4.3 SASS Security Context Release

5.4.3.1 Purpose/Description

The function is called to inform the application to release the specified security context. It must be implemented as a simple non-blocking function.

5.4.3.2 Prototype

```
*ScFree(sassHandle,  /* channel identifier */
        scID         /* security context ID */
        );
```

5.4.4 SASS Channel Routing Registration

5.4.4.1 Purpose/Description

The function is called for the application to register the specified software routing information into the PASS Lookup table. It may be triggered by the Sa_chanControl(), Sa_chanSendData() and Sa_chanReceiveData() APIs.

5.4.4.2 Prototype

```
*ChanRegister(sassHandle,   /*  channel identifier */
               swInfo       /* software routing information as defined at section 5.1.12 */
            );
```

5.4.5 SASS Channel Routing Un-registration

5.4.5.1 Purpose/Description

The function is called for the application to remove the specified software routing information from the PASS Lookup table. It may be triggered by the sSa_chanClose(), Sa_chanSendData() and Sa_chanReceiveData() APIs.

5.4.5.2 Prototype

```
*ChanUnregister(sassHandle,  /* channel identifier */
                 swInfo      /* software routing information as defined at section 5.1.12 */
            );
```

5.4.6 SASS Send Null Packet

5.4.6.1 Purpose/Description

The function is called to request the application to send the Null packet to the SA sub-system. The null packet is used to evict and/or tear down the security context associated with the channel. It may be triggered by the Sa_chanClose(), Sa_chanSendData() and Sa_chanReceiveData() APIs.

5.4.6.2 Prototype

```
*ChanSendNullPkt(sassHandle,  /* channel identifier */
                  pktInfo     /* packet information defined at section 5.1.14 */
            );
```

6 Multi Core Considerations

6.1 Multiple-Core System Resources

The SA LLD should work so that multiple instances of the driver can operate simultaneously on different cores. It is up to the application to maintain the multi-core resources such as the security context IDs and security context buffers which can be pre-allocated as per-core resources or be shared among multiple cores.

6.2 System Level Resources

Both the RNG (Random Number Generator) and the PKA (Public Key Accelerator) within SASS are shared among multiple cores. They can be invoked through the memory-mapped registers by one core at a time. Therefore, multiple-core semaphore or similar mechanism should be used to coordinate their usages.

6.3 System Level Operation

A coordination mechanism should be provided to make sure that all system level operations such as SASS reset, download and update will be performed on the master core only.

7 Design

7.1 GEM Software Resource Requirements

7.1.1 LLD MCPS Requirements

The per-channel MCPS requirements vary based on the channel protocol. The most MCPS-intensive operation will be the SRTP in pure software mode and it should be similar to the current MSU channel MCPS. The pre-channel MCPS in all other modes should be much smaller since all the MCPS-intensive encryption and authentication operations are performed by the SASS. The detailed LLD MCPS requirement will be provided in details when the software benchmark is available.

7.1.2 LLD Memory Requirements

7.1.2.1 Per System/Per IP block Instance Memory

- Protocol Specific Call Tables
- Encryption Mode Specific Mode Control Word Table (27 bytes per mode)
- Air Ciphering Mode Specific Mode Control Word Table (27 bytes per mode)

7.1.2.2 Per Channel/Connection/Application Link Memory Requirements

In addition to the channel instance memory which size is similar to the size of the MSU channel instance, each SASS channel needs to maintain two protocol-specific security contexts. The following table shows the SASS per-channel memory requirements:

Protocol	Instance	CTX Rx	CTX Tx	CTX Total
IPSEC AH	180	320	288	608
IPSEC ESP	180	320	288	608
SRTP	320	288	224	512
Air Ciphering	180	160	160	320

7.2 HW and PDSP Resource Requirements

7.2.1 Hardware/PDSP cycle information

All offloaded packets to the SASS will be divided into multiple data chunks of size 252 bytes or less. All data chunks will be processed by the packet header processing PDSP. It may enter the PHP PDSP multiple times. These run at 350 MHz, independent of the CPU clock (they are divided down from the SGMII PLL). The following table shows the maximum cycle requirements for different protocols:

Protocol	Maximum cycles per 64-byte packet
IPSEC AH	240
IPSEC ESP	260
SRTP	290
3GPP Air Ciphering	560

7.2.2 Hardware/PDSP memory information

PDSP memory is visible through the system configuration bus, but should not be accessed by the LLD or the application during normal operation.

7.3 Subsystem Integration Call Flow

The following sections illustrate the call flow for subsystem control and initialization, the SASS channel setup and packet data flow.

7.3.1 Subsystem Control/Initialization

Initialization is done by using the system APIs to put the subsystem into reset, download all required images, and release reset.

7.3.2 Channel Setup/Teardown Call flow

The CGEM application will invoke multiple SASS LLD API calls to setup and/or teardown a SASS security channel. Each SASS channel only represents the operation of a single security protocol. For the nested security channel such as SRTP over IPSEC, it is up to the application to setup and coordinate multiple SASS channels.

The following figures illustrate the typical call flow to setup and teardown a SASS channel.

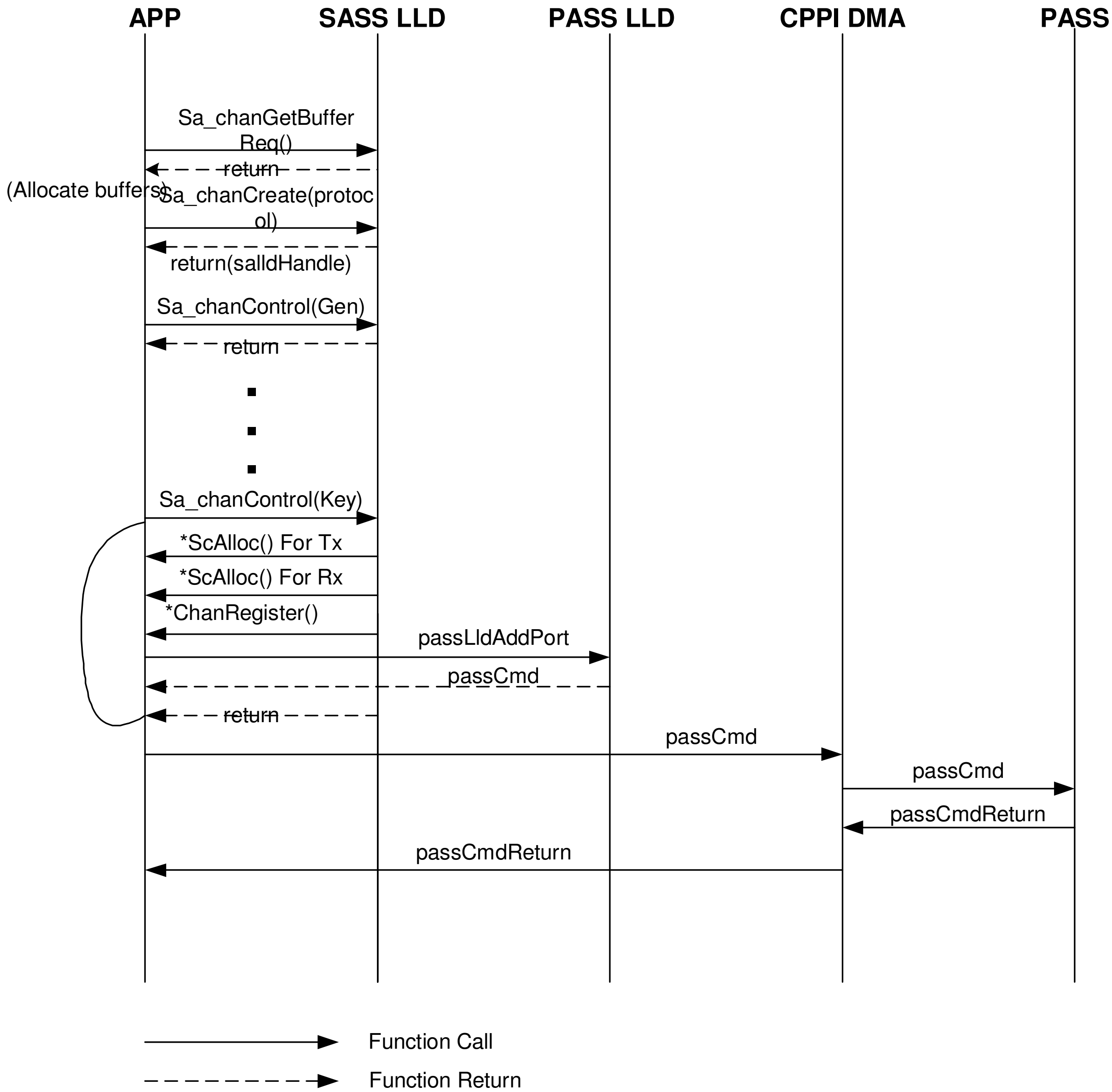

Figure 2. SASS Channel Setup Call Flow

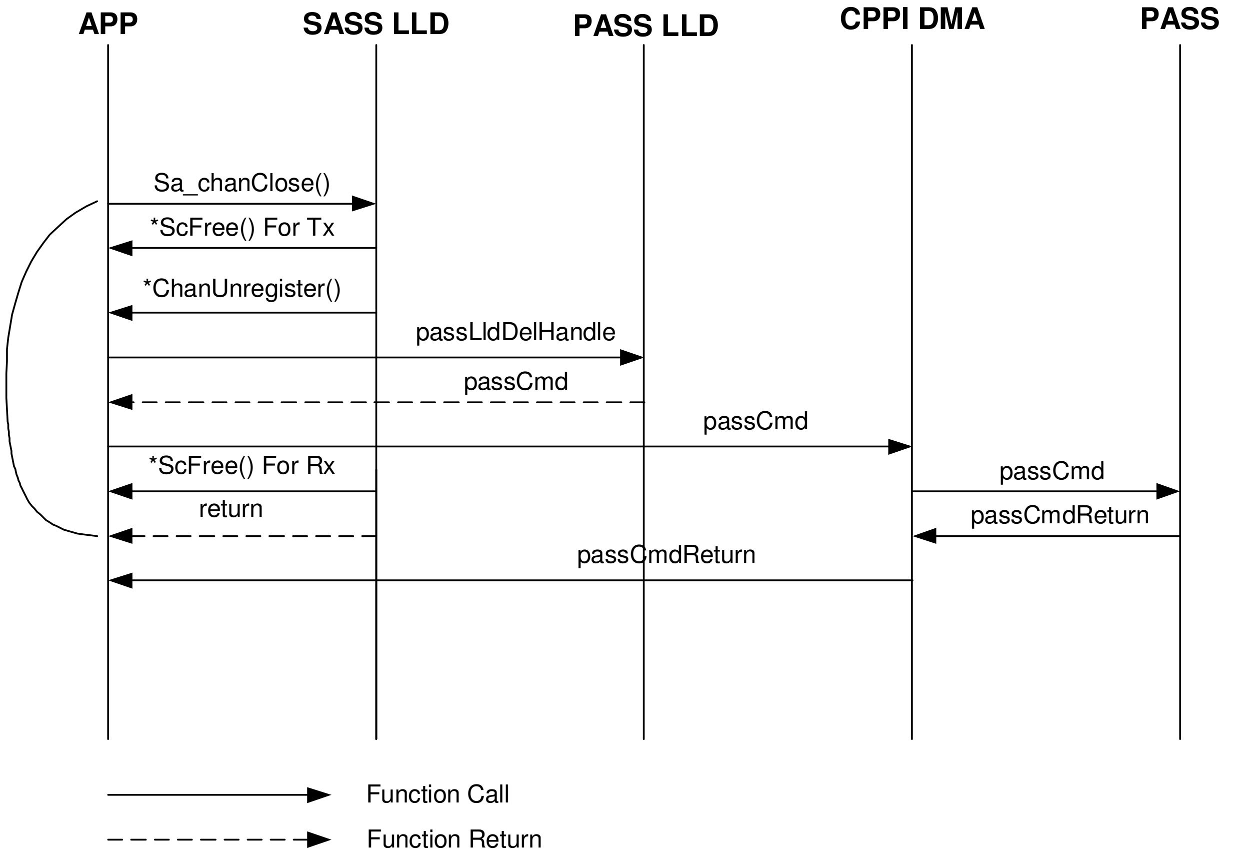

Figure 3. SASS Channel Teardown Call Flow

7.3.3 Data Flow

The application shall call the Sa_chanSendData() API to perform protocol-specific packet pre-processing in the To-Network direction. It needs to call the Sa_chanSendData() API for each security protocol for the nested security packet. It then prepares and forwards the packet to SASS through the CPPI DMA. Figure 4 shows the typical transmit data flow for SRTP over IPSEC packets.

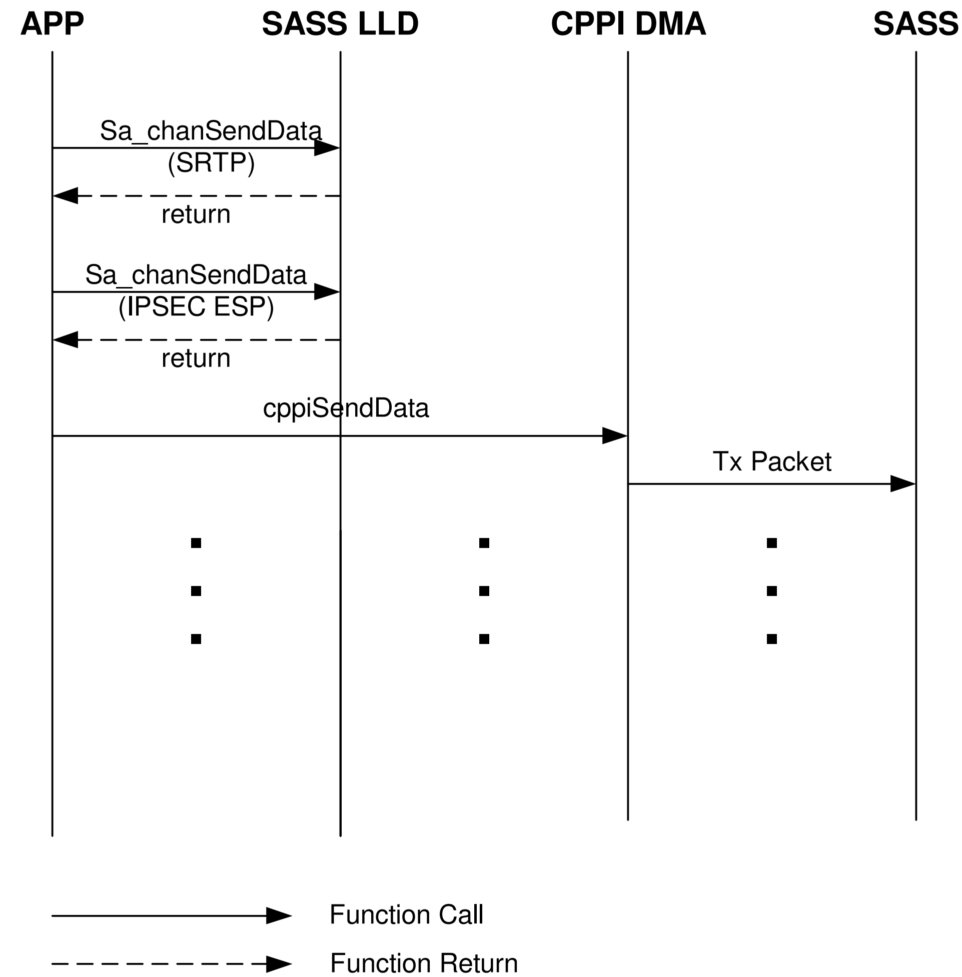

Figure 4. SASS Tx Data Flow (SRTP over IPSEC ESP)

The application shall call the Sa_chanReceiveData() API to perform protocol-specific packet post-processing when it receives the packet from the CPPI DMA in the From-Network direction. It may need to call the Sa_chanReceiveData() API for each security protocol for the nested security packet. It then continues the packet processing. Figure 5 shows the typical receive data flow for SRTP over IPSEC packets.

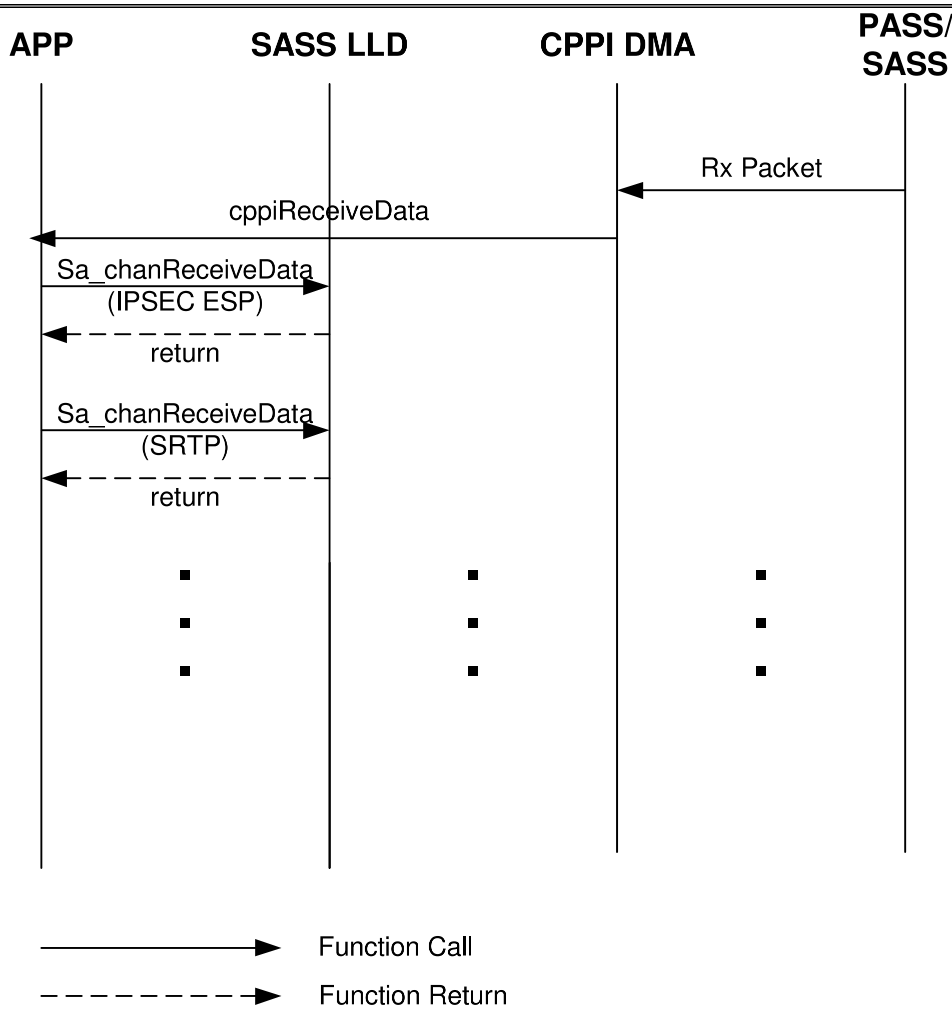

Figure 5. SASS Rx Data Flow (SRTP over IPSEC ESP)

8 Testing Considerations

Provide high level testing plans/ideas that will be later expanded into unit test plan and integration/system test plan.

Refer to the unit test plan.

9 Future Extensions

9.1 Known Issues

Put action items to be completed or known limitations to the design.

9.1.1 Multi-core Channel Support

It will be desired to support SA LLD channel over multiple DSP cores. New API such as global channel registration will be provided to support this feature.

9.2 Potential Enhancements

Put ideas that could be used to enhance future versions of the IP block.