

---

# SPECT Firmware

**API Documentation**

**Version: 0.1**

**Git tag: v0.3.0**

Tropic Square  
October 29, 2024





## Version history

Version Tag	Date	Author	Description
0.1	25.3.2024	Vit Masek	Initial version (copy from TROPIC01 functional specification)



---

## Contents

<b>1</b>	<b>SPECT Operations</b>	<b>3</b>
1.1	SPECT Application Firmware . . . . .	6
1.2	SPECT Boot Firmware . . . . .	12
1.3	SPECT Debug Operations . . . . .	15
1.4	Processing L3 Command packet using SPECT . . . . .	16
<b>2</b>	<b>Open Issues</b>	<b>18</b>

# 1 SPECT Operations

SPECT provides several operations for CPU that the CPU uses to perform more complex tasks, e. g. execute L3 Command packets.

SPECT comes with two types of firmware:

- **Boot FW** – small firmware to support boot-up sequence of the chip.
- **Application FW** – firmware that is preloaded into SPECTs instruction memory during boot-up sequence and used in final application.

## Note

Besides these, another set of SPECT operations is available for debug purposes. These operations do not rely on the functionality of SCB, CPB nor KBUS. EDB can be set to debug mode.

CPU specifies SPECT operation and SPECT firmware parameters by writing **SPECT\_CFG\_WORD** on address 0x0100 of SPECTs address space (Data RAM In).

## SPECT\_CFG\_WORD:

[31:16]	[15:8]	[7:0]
<b>SPECT_OP_DATA_IN_SIZE</b>	<b>SPECT_INOUT_SRC</b>	<b>SPECT_OP_ID</b>

CPU selects input data source via **SPECT\_INOUT\_SRC[7:4]**.

- **SPECT\_INOUT\_SRC[7:4]=0x0** – SPECT operation takes input data from its Data RAM In.
- **SPECT\_INOUT\_SRC[7:4]=0x4** – SPECT operation takes input data from Command Buffer.

CPU selects output data destination via **SPECT\_INOUT\_SRC[3:0]**.

- **SPECT\_INOUT\_SRC[3:0]=0x1** – SPECT operation stores output data to its Data RAM Out.
- **SPECT\_INOUT\_SRC[3:0]=0x5** – SPECT operation stores output data to Result Buffer.

CPU indicates number of bytes in Command Buffer or Data RAM In via **SPECT\_OP\_DATA\_IN\_SIZE** field.

After SPECT finishes current operation, CPU reads **SPECT\_RES\_WORD** from address 0x1100 in SPECTs address space (Data RAM Out).



### SPECT\_RES\_WORD:

[31:16]	[15:8]	[7:0]
SPECT_OP_DATA_OUT_SIZE	–	SPECT_OP_STATUS

SPECT indicates status of the executed operation via **SPECT\_OP\_STATUS**. SPECT also indicates number of bytes in Data RAM Out or Result Buffer via **SPECT\_OP\_DATA\_OUT\_SIZE**.

#### Note

When SPECT executes operation that corresponds to one whole L3 Command packet (L3 Command packet is not split into multiple SPECT operations), then **SPECT\_OP\_DATA\_IN\_SIZE = CMD\_SIZE** and **SPECT\_OP\_DATA\_OUT\_SIZE = RES\_SIZE**.

Values of **SPECT\_OP\_STATUS** after SPECT operation execution can be following:

Value	Description
0x00	SPECT Operation executed successfully
0xF1	SPECT Context error
0xF2	SPECT KBUS error
0xF3	SPECT Invalid OP_ID error
0xF4	SPECT Invalid Curve Type error
0xF5	SPECT GRV error
0xF6	SPECT Invalid ECC slot metadata error
0x11	X25519 Error: invalid priv key
0x12	X25519 Error: invalid pub key
0x21	ECDSA Error: invalid ECDSA nonce ( $k \equiv 0 \pmod{q_{P256}}$ )
0x22	ECDSA Error: invalid ECDSA $r$ ( $r \equiv 0 \pmod{q_{P256}}$ )
0x23	ECDSA Error: invalid ECDSA $s$ ( $s \equiv 0 \pmod{q_{P256}}$ )
0x24	ECDSA Error: final verify fail
0x34	EdDSA Error: invalid priv key $s$
0x35	EdDSA Error: invalid pub key $A$
0x36	EdDSA Error: final verify fail
0x41	Point Integrity Error

To perform some more complex algorithm, such as EdDSA or secure channel establishment, CPU needs to invoke SPECT multiple times. In such a case, SPECT keeps context between execution of each partial operation.

SPECT Context is a set of values (e.g. keys or context of TMAC), that SPECT keeps inside and uses them during its next operations. CPU shall invoke only such SPECT operation, which input context matches the output context (or part of it) of previously invoked SPECT operation. Otherwise the context would be corrupted or the result of such SPECT



operation undefined.

SPECT is capable of processing these L3 Command packets by it self:

- ***ECC\_Key\_Generate*** – ***ecc\_key\_generate*** SPECT Op
- ***ECC\_Key\_Store*** – ***ecc\_key\_store*** SPECT Op
- ***ECC\_Key\_Read*** – ***ecc\_key\_read*** SPECT Op
- ***ECC\_Key\_Erase*** – ***ecc\_key\_erase*** SPECT Op
- ***ECDSA\_Sign*** – ***ecdsa\_sign*** SPECT Op

#### Note

**SPECT\_INOUT\_SRC** is used for debug purpose, when unencrypted L3 Command packet is send to the device. That allows the CPU to bypass SCB and CPB and load the content of the L3 Command packet directly to SPECTs Data RAM In. In final implementation, no such bypass shall be possible.

## 1.1 SPECT Application Firmware

### Control Operations

<b>clear</b>	<b>SPECT_OP_ID:</b>	0x00
Clears all GPRs and Data RAM In/Out. Clears context of SHA512 and TMAC.		

### ECC Key Operations

Following operations corresponds to eponymous L3 Command packets. The value of each **SPECT\_OP\_ID** is equal to corresponding L3 Command packets **CMD\_ID**. The layout of inputs and outputs is also the same.

<b>ecc_key_generate</b>	<b>SPECT_OP_ID:</b>	0x60
Computes ECC keys for ECDSA/EdDSA based on seed from TRNG, and stores them to user slot $i$ of ECC Key partition in Flash memory.		
Command Buffer / Data RAM In		
0x00	1B	<b>CMD_ID</b>
0x01	2B	<b>SLOT</b> (Key slot $i$ )
0x03	1B	<b>CURVE</b> (Curve Type (P256 - ECDSA, Ed25519 - EdDSA))
Result Buffer / Data RAM Out		
0x00	1B	<b>RESULT</b>

<b>ecc_key_store</b>	<b>SPECT_OP_ID:</b>	0x61
Computes ECC keys for ECDSA/EdDSA based on seed $k$ , and stores them to user slot $i$ of ECC Key partition in Flash memory.		
Command Buffer / Data RAM In		
0x00	1B	<b>CMD_ID</b>
0x01	2B	<b>SLOT</b> (Key slot $i$ )
0x03	1B	<b>CURVE</b> (Curve Type (P256 - ECDSA, Ed25519 - EdDSA))
0x10	32B	<b>K</b> ( $k$ )
Result Buffer / Data RAM Out		
0x00	1B	<b>RESULT</b>



ecc_key_read		SPECT_OP_ID:	0x62
Reads ECDSA/EdDSA public key $A$ from slot $i$ of ECC Public Key partition.			
Command Buffer / Data RAM In			
0x00	1B	CMD_ID	
0x01	2B	SLOT (Key slot $i$ )	
Result Buffer / Data RAM Out			
0x00	1B	RESULT	
0x01	1B	CURVE (Curve Type (P256, Ed25519))	
0x02	1B	ORIGIN (Identifier weather the key was stored or generated.)	
0x10	64/32B	PUB_KEY (ECDSA/EdDSA ECC Pub Key $A$ )	

ecc_key_erase		SPECT_OP_ID:	0x63
Erase ECDSA/EdDSA keys from slot $i$ of ECC Public/Private Key partition.			
Command Buffer / Data RAM In			
0x00	1B	CMD_ID	
0x01	2B	SLOT (Key slot $i$ )	
Result Buffer / Data RAM Out			
0x00	1B	RESULT	





## Secure Channel Handshake Operations

All SPECT operations in this section are supported only with use of Data RAM In / Out (**SPECT\_INOUT\_SRC**=0x10).

x25519_kpair_gen			SPECT_OP_ID:	0x11
Generates ephemeral X25519 key pair used during secure channel handshake.				
Data RAM Out				
0x1020	32B	$E_{TPUB}$		
Context In			Context Out	
None			{ $E_{TPRIV}$ }	

x25519_sc_et_eh			SPECT_OP_ID:	0x12
Calculates $X1 = X25519(E_{TPRIV}, E_{HPUB})$ .				
Data RAM In				
0x0020	32B	$E_{HPUB}$		
Data RAM Out				
0x1020	32B	$R1$		
Context In			Context Out	
$\{E_{TPRIV}\}$			$\{E_{TPRIV}, E_{HPUB}\}$	

x25519_sc_et_sh		SPECT_OP_ID:	0x13
Calculates $R2 = X25519(E_{TPRIV}, S_{HIPUB})$ .			
Data RAM In			
0x0020	1B	Key slot $i$	
Data RAM Out			
0x1020	32B	$R2$	
Context In		Context Out	
$\{E_{TPRIV}, E_{HPUB}\}$		$\{E_{HPUB}\}$	

x25519_sc_st_eh			SPECT_OP_ID:	0x14
Calculates $R3 = X25519(S_{TPRIV}, E_{HPUB})$ .				
Data RAM Out				
0x1020	32B	$R3$		
Context In			Context Out	
{ $E_{HPUB}$ }			None	



## EdDSA Operations

eddsa_set_context		SPECT_OP_ID:	0x41
Sets context for Edwards Digital Signature Algorithm such as keys ( $s, prefix, A$ ) and Secure Channel data ( $SC_H, SC_N$ ).			
Command Buffer / Data RAM In			
0x01	2B	Key slot $i$	
Data RAM In			
0x00A0	32B	Secure Channel Hash $SC_H$	
0x00C0	4B	Secure Channel Nonce $SC_N$	
Context In		Context Out	
None		{keys, $SC_N, SC_H$ }	

eddsa_nonce_init			SPECT_OP_ID:	0x42
Initialize EdDSA nonce $r$ derivation – $\text{TMAC\_INIT}(prefix, 0x0C)$ .				
Context In			Context Out	
{keys, $SC_N, SC_H$ }			{keys, TMAC ctx}	

eddsa_nonce_update			SPECT_OP_ID:	0x43
Processes next 144 bytes of decrypted message M. Updates TMAC context.				
Command Buffer / Data RAM In				
0x00	144B	Next 144 bytes of decrypted message M		
Context In			Context Out	
{keys, TMAC ctx}			{keys, TMAC ctx}	

eddsa_nonce_finish			SPECT_OP_ID:	0x44
Finishes the EdDSA nonce derivation: $r = \text{TMAC}(prefix, SC_H    SC_N    M, 0x0C) \pmod{q_{Ed25519}}$ .				
Command Buffer / Data RAM In				
0x00	–	Last up to 144 bytes of decrypted message M		
Context In			Context Out	
{keys, TMAC ctx, part of M}			{keys, $r$ , M size}	



<b>eddsa_R_part</b>	<b>SPECT_OP_ID:</b>	0x45
Computes part $R$ of the signature – $R = r \cdot G_{Ed25519}$ .		
Context In	Context Out	
{keys, $r$ , M size}	{keys, $R$ }	

<b>eddsa_e_at_once</b>	<b>SPECT_OP_ID:</b>	0x46
Computes $e$ at once in case of size of the message is less 64 bytes.		
Command Buffer / Data RAM In		
0x00	–	First up to 63 bytes of decrypted message M
Context In	Context Out	
{keys, $R$ }	{keys, $R$ , $e$ }	

<b>eddsa_e_prep</b>	<b>SPECT_OP_ID:</b>	0x47
Prepares for derivation of $e$ .		
Command Buffer / Data RAM In		
0x00	64	First 64 bytes of decrypted message M
Context In	Context Out	
{keys, $R$ }	{keys, $R$ , SHA512 ctx}	

<b>eddsa_e_update</b>	<b>SPECT_OP_ID:</b>	0x48
Processes next 128 bytes of decrypted message M. Updates SHA512 context.		
Command Buffer / Data RAM In		
0x00	128B	Next 128 bytes of decrypted message M
Context In	Context Out	
{keys, $R$ , SHA512 ctx}	{keys, $R$ , SHA512 ctx}	

<b>eddsa_e_finish</b>	<b>SPECT_OP_ID:</b>	0x49
Finishes derivation of $e = \text{SHA512}(R    A    M)$ .		
Command Buffer / Data RAM In		
0x00	–	Last up to 128 bytes of decrypted message M
Context In	Context Out	
{keys, $R$ , SHA512 ctx}	{keys, $R$ , $e$ }	



eddsa_finish		SPECT_OP_ID:	0x4A
Computes second part $S$ of the signature – $S = r + e \times s \pmod{q_{Ed25519}}$			
Result Buffer / Data RAM Out			
0x00	1B	<b>RESULT</b>	
0x10	64B	Signature ( <b>R</b> , <b>S</b> )	
Context In		Context Out	
{keys, $R$ , $e$ }		None	

## ECDSA Operations

ecdsa_sign		SPECT_OP_ID:	0x70
Generate ECDSA signature ( $r, s$ ) of a message digest $z$ using ECC keys from slot $i$ .			
Command Buffer / Data RAM In			
0x01	2B	<b>SLOT</b> (ECC Key slot $i$ )	
0x10	32B	<b>MSG_HASH</b> (Message digest $z$ )	
Data RAM In			
0x00A0	32B	Secure Channel Hash $SC_h$	
0x00C0	4B	Secure Channel Nonce $SC_n$	
Result Buffer / Data RAM Out			
0x0	1B	<b>RESULT</b> (Command Result)	
0x10	64B	Signature ( <b>R</b> , <b>S</b> )	

### Note

eddsa\_set\_context and ecdsa\_sign SPECT operation takes Secure Channel Handshake Hash ( $SC_h$ ) and Secure Channel Session Nonce ( $SC_n$ ) always from Data RAM In, never from Command Buffer regardless of the **SPECT\_INOUT\_SRC** value.



## 1.2 SPECT Boot Firmware

### EdDSA Verify

eddsa_verify		SPECT_OP_ID:	0x4B
Verifies a EdDSA signature of 32-byte message $M$ using public key $A$ .			
Data RAM In			
0x0020	64B	Signature ( $\mathbf{R}$ , $\mathbf{S}$ )	
0x0060	32B	Public key $A$	
0x0080	32B	Message $M$	
Data RAM Out			
0x0000	4B	Result: 0x0B5E55ED $\Rightarrow$ verified, else failed	
0x0004	4B	Result: 0xBA11FADE $\Rightarrow$ verified, else failed	

### SHA512 Operations

Removed since SPECT FW v0.3.0

sha512_init		SPECT_OP_ID:	0x51
Initialize SHA512 unit.			
Context In		Context Out	
None		SHA512 ctx	

Removed since SPECT FW v0.3.0

sha512_update		SPECT_OP_ID:	0x52
Process one 128B block of data with SHA512.			
Data RAM In			
0x10	128B	Data	
Context In		Context Out	
SHA512 ctx		SHA512 ctx	

Removed since SPECT FW v0.3.0

sha512_final		SPECT_OP_ID:	0x53
Process last 128B block of data that contains SHA512 padding.			
Data RAM In			
0x10	128B	Data	
Data RAM Out			
0x10	64B	Digest	
Context In		Context Out	



SHA512 ctx	None
------------	------



### SHA512 example:

Let **M** be a 326 byte data (`u8 M [326]`). SHA512(**M**) is computed as follows:

1. Pad the data with SHA512 padding to multiple of 128B, see 1.
2. Initialize SPECTs SHA512 unit with **sha512\_init()**.
3. Process first chunk with **sha512\_update(M[0:127])**.
4. Process second chunk with **sha512\_update(M[128:255])**.
5. Process the rest of **M** with **sha512\_final(M[256:383])**.
6. Data RAM Out now contains the SHA512 digest of **M**.

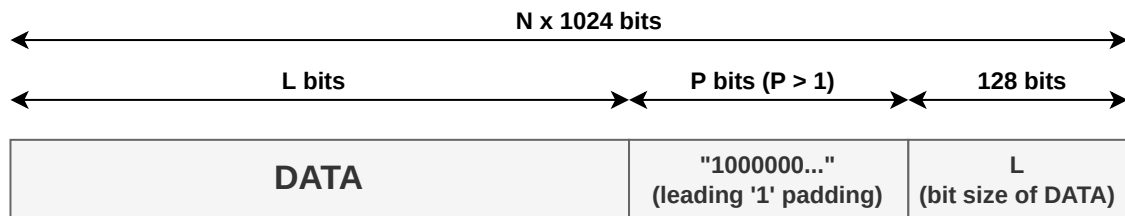


Figure 1: SHA512 Padding



### 1.3 SPECT Debug Operations

All SPECT operations in this section are supported only with use of Data RAM In / Out (**SPECT\_INOUT\_SRC=0x01**).

x25519_dbg		SPECT_OP_ID:	0x9F
Performs general X25519 algorithm. $R = \text{X25519}(\text{Priv}, \text{Pub})$			
Data RAM In			
0x0020	32B	Private key $\text{Priv}$	
0x0040	32B	Public key $\text{Pub}$	
Data RAM Out			
0x1020	32B	Result $R$	

ecdsa_sign_dbg		SPECT_OP_ID:	0xAF
Computes signature of message digest $z$ using ECDSA.			
Data RAM In			
0x0010	32B	Message digest $z$	
0x0040	64B	Private key pair $(d, w)$	
0x00A0	32B	Secure Channel Hash $SC_H$	
0x00C0	4B	Secure Channel Nonce $SC_N$	
0x0160	64B	Public key $(A_x, A_y)$	
Data RAM Out			
0x1010	64B	Signature	

eddsa_set_context_dbg			SPECT_OP_ID:	0xBF
Sets context for EdDSA sequence from Data RAM In only.				
Data RAM In				
0x0040	64B	Private keys $(s, \text{prefix})$		
0x00A0	32B	Secure Channel Hash $SC_H$		
0x00C0	4B	Secure Channel Nonce $SC_N$		
0x0300	32B	Public key $A$		
Context In			Context Out	
None			{keys, $SC_N$ , $SC_H$ }	





## 1.4 Processing L3 Command packet using SPECT

CPU always issues first 16 byte chunk of L3 Command packets **CMD\_CIPHERTEXT** to SCB. SCB decrypts the chunk and passes it to CPB. CPB checks user access privileges and stores the decrypted **CMD\_CIPHERTEXT** into Command Buffer. Then CPU reads the L3 Command packets **CMD\_ID** from CPB. Following three cases may occur:

### **CMD\_ID corresponds to one of *ECC\_Key\_\****

In this case, CPU issues the rest of **CMD\_CIPHERTEXT** to SCB. After that, Command Buffer contains all data and information needed for execution of the corresponding SPECT operation. CPU sets **SPECT\_INOUT\_SRC=0x45** and invokes SPECT.

SPECT reads Command Buffer and performs the corresponding SPECT operation. Then it stores all output data for the L3 Result packet to Result Buffer, sets **SPECT\_OP\_DATA\_OUT\_SIZE** to number of bytes it has stored and notifies CPU.

CPU reads **SPECT\_OP\_DATA\_OUT\_SIZE** bytes from Result Buffer, that are encrypted by SCB resulting in **RES\_CIPHERTEXT** of the L3 Result packet.

### **CMD\_ID corresponds to *ECDSA\_Sign***

In this case, the process is almost the same as in the case of ***ECC\_Key\_\****. In addition, CPU reads the Secure Channel Nonce and Secure Channel Hash from SCB and loads it in to SPECT's Data RAM In before invoking SPECT.

### **CMD\_ID corresponds to *EDDSA\_Sign***

In this case, the L3 Command packet has arbitrary length, depending on the length of the **MSG** field. Therefore the ***EDDSA\_Sign*** L3 Command packet is processed with a sequence of SPECT operations from 1.1. We can divide this sequence into 5 phases:

1. Setting context and loading keys – **eddsa\_set\_context / eddsa\_set\_context\_dbg**
2. Computing nonce  $r$  using TMAC – **eddsa\_nonce\_\***
3. Computing signature part **R** – **eddsa\_R\_part**
4. Computing  $e = \text{SHA512}(\mathbf{R}, A, \mathbf{MSG})$  – **eddsa\_e\_\***
5. Computing signature part **S** – **eddsa\_finish**

Figure 2 shows the sequence in detail.

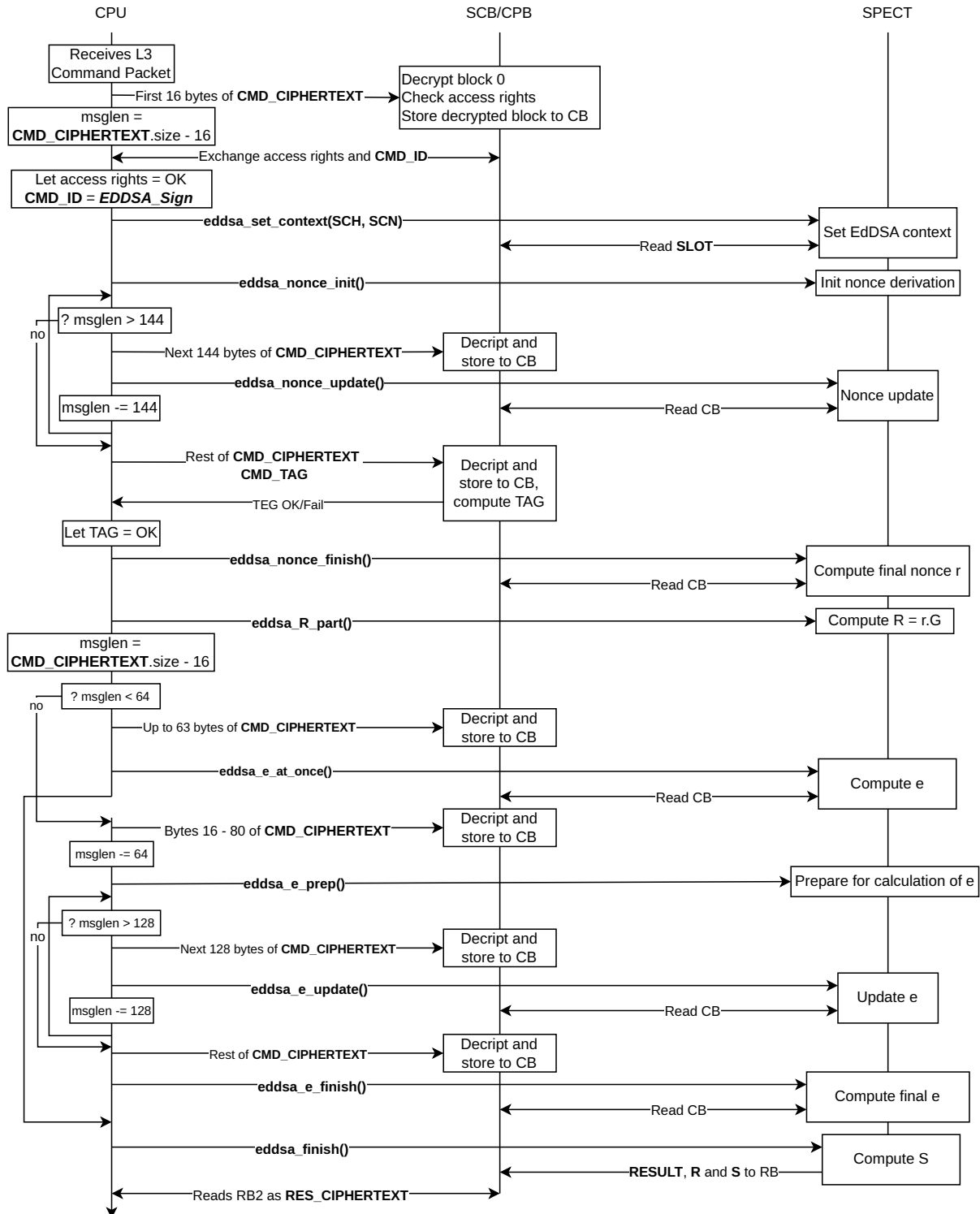


Figure 2: SPECT Firmware - EDDSA Data Flow



## 2 Open Issues

Document contains following open issues: