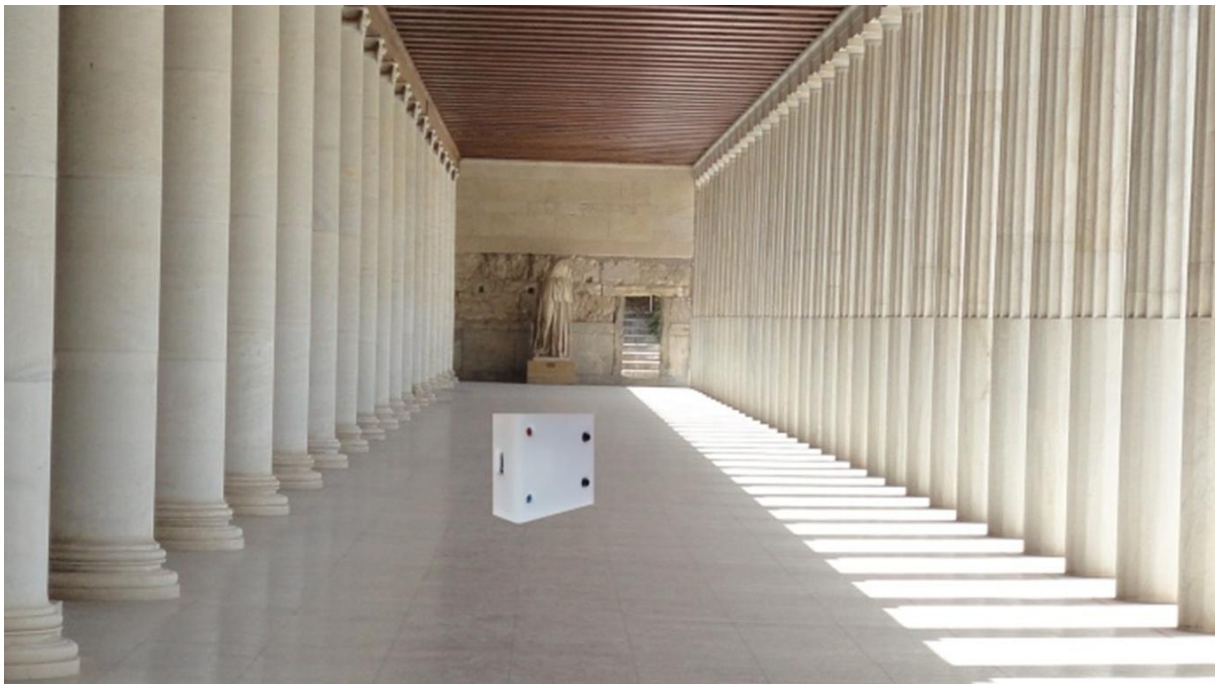
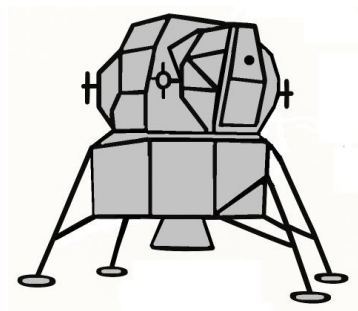


LeMonolith

A New Paradigm for Secure Elements



LeMonolith Dev Kit (LEM) v0.8



1	FOREWORD	9
2	ABOUT LEMONOLITH	10
3	INITIALIZATION OF THE LEM DEVKIT	11
4	LOADING SOFTWARE.....	12
4.1	ESP32 software loader.....	12
4.2	Factory operations	12
4.3	Secure Element Software Loader	12
4.3.1	Load applications into the secure element.....	13
4.3.2	List applications stored in the secure element	13
5	OPERATING MODE SELECTION.....	14
5.1	Using push buttons.....	14
5.2	Using Serial Terminal	14
5.3	Setting Wi-Fi parameters in Bluetooth mode	16
6	OVERVIEW	17
6.1	USB Mode	17
6.2	USB Bluetooth mode	18
6.3	Wi-Fi Mode	18
6.3.1	OPENSSL.....	18
6.3.2	WolfSSL	18
6.3.3	EtherSSL	19
6.3.4	Command list for the TLSSE client	19
6.3.5	Using multiple identities	21
6.3.6	File encryption	21
6.4	Wi-Fi TLS-IM Mode	23
6.4.1	Example use of a remote smartcard reader	24
6.5	TLS Identity Module.....	25
6.6	Recursive Authentication with TLSSE client	25
6.7	IoSE server for Wi-Fi mode.....	27
6.8	Bluetooth Mode	28
7	LEMONOLITH QUICK TESTING	28
7.1	openssl.bat, WolfSSL.bat	28

7.2	openssl_AESGCM.bat, WolfSSL_AESGCM.bat	28
7.3	openssl_PKI.bat	28
7.4	ListSE.bat	28
7.5	DeleteSE.bat	28
7.6	LoadSE.bat	29
7.7	LoadSE2.bat	29
7.8	sign.bat	29
7.9	auth.bat	29
7.10	TLS13_Server_PSK_AESCCM.bat	29
7.11	connect_127_0_0_1_444_im_pcsc.bat	29
7.12	reader.exe, winsreader.exe	29
8	USB COMMAND SHELL	30
9	USB BLUETOOTH COMMAND SHELL	30
10	TLS-IM WI-FI COMMAND SHELL	33
11	WI-FI OPERATIONS	33
11.1	Example OpenSSL command line	34
12	TLS-SE VERSION 3.0.5	34
12.1	Command Formats	35
12.1.1	Request	35
12.1.2	Response	35
12.2	Text Format	35
12.2.1	Request	35
12.2.2	Response	36
12.3	Binary Format	36
12.3.1	Request	36
12.3.2	Response	36
12.4	Command Index Encoding	36
12.4.1	Administrative Command	36
12.4.2	Cryptographic Command	36
12.4.3	Read/Write Command	36
12.5	Blobs and Wrap Secrets	37
12.5.1	Blob Format	37
12.5.1.1	BlobType	37

12.5.1.2	KeyIndex	37
12.5.1.3	KeyParam.....	37
12.5.1.4	BlobRnd	37
12.5.1.5	BlobID	37
12.5.1.6	EncryptedValue	37
12.5.1.7	Tag	37
12.5.2	Blob Computation	38
12.6	TLS-SE Components.....	38
12.6.1	Keys	38
12.6.2	Secrets.....	38
12.6.3	Wrap Secrets	38
12.6.4	Export/Import Keys	38
12.6.5	Files	38
12.7	Performance	38
12.8	TLS-SE App Commands, Version 3.05 (tls_se_305.cap)	39
12.9	Special Operation Mode (som)	41
12.10	Access Policy	42
12.11	TLS-SE Application Certification Procedure (ACP).....	42
12.12	TLS-SE Session Authentication Procedure (SAP)	43
12.13	TRAIT command.....	43
12.14	TLS-SE-IO commands.....	43
13	BLUETOOTH OPERATIONS.....	44
13.1	Serial Bluetooth terminal	45
13.2	Bluetooth CryptoToken App for Android	46
14	BLUETOOTH TLS-PSK (BTPSK).....	47
14.1	Testing Bluetooth TLS-PSK.....	47
15	LEMONOLITH (LEM) DEV KIT TESTS	48
15.1	USB Operations.....	48
15.1.1	COM_List.bat.....	48
15.1.2	COM_Find.bat	48
15.1.3	TERM_hyperterminal.bat	48
15.1.4	TERM_terminal.bat	48
15.1.5	USB_GP_list.bat.....	48
15.1.6	USB_GP_delete.bat	48
15.1.7	USB_GP_install.bat.....	48
15.1.8	USB_KEYSTORE_Genkey00.bat	48
15.1.9	USB_SERVER_TLSSE.bat	48
15.2	Wi-Fi Operations	48

15.2.1	SSL_openssl.bat.....	49
15.2.2	SSL_openssl_trait.bat.....	49
15.2.3	SSL_wolfssl.bat.....	49
15.2.4	SSL_wolfssl_test.bat.....	49
15.2.5	SSL_openssl_guest.bat.....	49
15.2.6	SSL_wolfssl_guest.bat.....	49
15.2.7	SSL_wolfssl_MFA.bat.....	49
15.2.8	SSL_wolfssl_PCSC.bat.....	49
15.2.9	KEYSTORE_NET_Load_Key.bat.....	49
15.2.10	KEYSTORE_NET_Load_Key_SC.bat.....	49
15.2.11	KEYSTORE_NET_Load_Key_MFA.bat.....	49
15.2.12	KEYSTORE_NET_test_sign.bat.....	49
15.2.13	CIPHER_NET_init.bat.....	50
15.2.14	CIPHER_NET_encrypt.bat.....	50
15.2.15	CIPHER_NET_decrypt.bat.....	50
15.3	Wi-Fi Operations with TLS-IM.....	50
15.3.1	TLSIM_GP_USB_LOADER_IM.bat.....	50
15.3.2	TLSIM_GP_USB_DELETE_IM.bat.....	50
15.3.3	TLSIM_GP_USB_PERSO_IM.bat.....	50
15.3.4	TLSIM_GP_USB_LOADER_IM0.bat.....	50
15.3.5	TLSIM_GP_USB_DELETE_IM0.bat.....	50
15.3.6	TLSIM_GP_USB_PERSO_IM0.bat.....	50
15.3.7	TLS_IM_NET_GP_LIST.bat.....	50
15.3.8	TLS_IM_NET_GP_DELETE.bat.....	50
15.3.9	TLS_IM_NET_GP_LOAD.bat.....	50
15.3.10	TLS_IM_NET_GP_LOAD2.bat.....	50
15.3.11	TLSIM_LEM_Client_PSK_AESGCM_reader.bat.....	50
15.3.12	TLSIM_LEM_Client_PKI_AESGCM_echo.bat.....	51
15.3.13	TLSIM_LEM_Client_PSK_AESCCM_tlsse.bat.....	51
15.3.14	TLSIM_SERVER_LOCAL_PSK_AESCCM.bat.....	51
15.3.15	TLSIM_SERVER_LEM_CONNECT_REMOTE.....	51
15.3.16	TLSIM_SERVER_LEM_CONNECT_PCSC.bat.....	51
15.3.17	TLSIM_SERVER_LEM_CONNECT_SERIAL.bat.....	51
15.4	USB BLUETOOTH Tests	51
15.5	TLS_SE_305 Tests	51
16	SECURE ELEMENT CERTIFICATION PROCEDURE OVER WI-FI	52
16.1	Loading the certification authority key	52
16.1.1	TLS-IM Smartcard	52
16.1.2	TLS-IM MFA Token	52
16.2	SE_NET_Cert_SOFT.bat	52
16.3	SE_NET_Cert_SC.bat.....	52
16.4	SE_NET_Cert_MFA.bat	52
17	SE AUTHENTICATION SESSION PROCEDURE (ASP) OVER WI-FI	52
17.1	SE_NET_auth_SOFT.bat.....	52
17.2	SE_NET_auth_SC.bat.....	52

17.3	SE_NET_auth_MFA.bat	52
18	IOSE TESTS	52
18.1	IOSE_Server_WIN32.bat.....	53
18.2	IOSE_Server_Console.bat	53
18.3	IOSE_RACS_List.bat.....	53
18.4	IOSE_RACS_Console	53
18.5	IOSE_GP_list.bat	53
18.6	IOSE_GP_delete	53
18.7	IOSE_GP_install.....	53
18.8	IOSE_Openssl.bat.....	53
18.9	IOSE_KEYSTORE_test_sign.bat	53
18.10	IOSE_Cert_SOFT.bat.....	53
18.11	IOSE_Cert_SC.bat	53
18.12	IOSE_Cert_MFA.bat	53
18.13	IOSE_auth_SOFT.bat	53
18.14	IOSE_auth_SC.bat	53
18.15	IOSE_auth_MFA.bat.....	54
19	ETHEREUM TRANSACTIONS OVER WI-FI	54
19.1	Ethereum transaction parameters.....	54
19.2	ETH_gasview.bat.....	54
19.3	ETH_NET_Make_Transaction.bat.....	54
19.4	ETH_Transaction_Send.bat.....	54
19.5	ETH_Transaction_View.bat	54
20	CERTIFICATE GENERATION	54
20.1	The fake private key technique	54
20.2	CERT_NET_MAKECA.bat.....	55
20.3	CERT_NET_MAKECSR.bat	55
20.4	CERT_NET_MAKECSR_TLSSE.....	55

20.5	CERT_NET_MAKECERT.bat	55
20.6	CERT_NET_MAKECERT2.bat.....	55
21	SOFTWARE.....	56
21.1	Software components	56
21.2	How to build LeMonolith.....	56
22	ONLINE TECHNICAL RESOURCES.....	56
22.1	TLS for Secure Element, TLS-SE	56
22.2	TLS for secure element input output TLS-SE-IO	56
22.3	TLS identity module, TLS-IM	56
22.4	Remote APDU Server (RACS)	56
22.5	TLS for secure element Rendez-Vous TLS-SE-RDV.....	56
22.6	Remote APDU Call Secure Lite(RACSL)	56
23	ANNEXE	57
23.1	The Crypto Currency (CC) Application	57
23.1.1	The Select Command	57
23.1.2	The Verify UserPin Command	57
23.1.3	The Verify UserPin2 Command	58
23.1.4	The Verify AdminPin command.....	58
23.1.5	The ChangePin command.....	59
23.1.6	The GetStatus command.....	59
23.1.7	The Write Command	60
23.1.8	The Read Command	60
23.1.9	The Clear KeyPair & InitCurve Command.....	61
23.1.10	The InitCurve & InitTree Command	61
23.1.11	The Generate KeyPair Command	62
23.1.12	The Dump KeyPair Command	63
23.1.13	The GetInfo command	64
23.1.14	The Get KeyParameter Command.....	65
23.1.15	The Set KeyParameter Command	66
23.1.16	The SignECDSA command	67
23.1.17	The GetCertificate command	68
23.1.18	The SetCertificate command.....	68
23.2	The TLS-IM Application	69
23.2.1	The Select Command	69
23.2.2	The Verify UserPin Command	69
23.2.3	The Verify AdminPin command.....	70
23.2.4	The ChangePin command.....	70
23.2.5	The GetStatus command.....	70
23.2.6	The Write Command	71
23.2.7	The Read Command	71
23.2.8	The Clear KeyPair command	72

23.2.9	The InitCurve command	72
23.2.10	The Generate KeyPair Command	73
23.2.11	The Get KeyParameter Command.....	73
23.2.12	The Set KeyParameter Command	74
23.2.13	The SignECDSA command	75
23.2.14	The Diffie Hellman Command	75
23.2.15	The GenerateRandom command	76
23.2.16	The HMAC Command	76
23.2.17	The GetCertificate command	77
23.2.18	The SetCertificate command.....	77
23.3	The TLS-IM0 Application.....	78
23.3.1	The Select Command	78
23.3.2	The Verify UserPin Command	78
23.3.3	The Verify AdminPin command.....	78
23.3.4	The ChangePin command.....	79
23.3.5	The HMAC Command	79
23.4	The TLS-SE Strict Application	81
23.4.1	The Select Command	81
23.4.2	The SEND Command	81
23.4.3	The RECV Command.....	82
23.5	The TLS-SE Combi Application	83
23.5.1	The Select Command	83
23.5.2	The Verify UserPin Command	83
23.5.3	The Verify AdminPin command.....	83
23.5.4	The ChangePin command.....	84
23.5.5	The GetStatus command.....	84
23.5.6	The Write Command	85
23.5.7	The Read Command	85
23.5.8	The Clear KeyPair command	86
23.5.9	The InitCurve command	86
23.5.10	The Generate KeyPair Command	87
23.5.11	The Get KeyParameter Command.....	87
23.5.12	The SetKeyParameter EC256 Command	89
23.5.13	The SetKeyParameter RSA2048 Command	90
23.5.14	The SignECDSA command	90
23.5.15	The ECDHE command.....	91
23.5.16	The HMAC Command	92
23.5.17	The GenerateRandom command	93
23.5.18	The GetCertificate command	93
23.5.19	The SetCertificate command.....	94
23.5.20	The SEND Command	94
23.5.21	The RECV Command.....	95
23.5.22	The TEST command	95

1 Foreword

In an increasingly digital world, organizations and individuals need to protect their digital assets. Encryption provides this protection and relies on cryptographic keys. If a hacker gains access to your encryption keys, you lose that protection. Securing these keys is therefore a primary objective. Organizations use hardware security modules (HSMs), hardened against intrusion and tampering, to store and provision cryptographic keys for encryption, decryption, and authentication. HSMs are expensive to purchase and manage. LeMonolith offers better security at a fraction of the price, in a form factor suitable for individual use, while significantly reducing operating costs. Housed in a cube the size of a matchbox, it combines the ubiquitous secure element (the chip used in bank cards), thoroughly vetted open-source security middleware, Bluetooth, and Wi-Fi for easy integration into your home or office environment.

LeMonolith is a personal network HSM (pNHSM) accessible over Wi-Fi. It also serves as:

- ✓ A serial USB smartcard reader.
- ✓ A network smartcard reader over TLS.
- ✓ A smartcard reader over Bluetooth.
- ✓ A TLS-PSK server over Bluetooth.

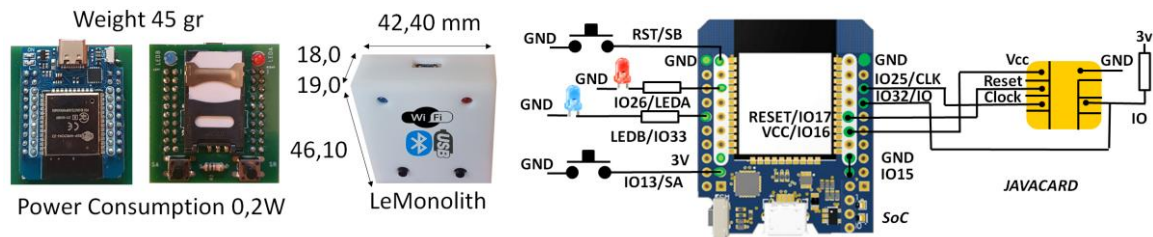
LeMonolith uses open hardware and software platforms:

- ✓ The software development environment uses the *Arduino Integrated Development Environment* (IDE) and the free *Oracle Java Card Development Kit* (JDK).
- ✓ The hardware consists of an ESP32, a well-known system on chip (SoC), and a Java Card, a secure device widely used in digital technologies.

The LeMonolith development kit supports three main security services:

- ✓ Ethereum transaction generation (with the SEPOLIA test blockchain).
- ✓ File Encryption/Decryption.
- ✓ Certificate generation with OpenSSL.

2 About LeMonolith



LeMonolith has two main components: an ESP32 D1 Mini board and a Java Card. It demonstrates security microservers based on secure elements. An introduction to an online TLS-SE secure element is available on YouTube:

see <https://www.YouTube.com/watch?v=0cLtrcMNjQ4>

LeMonolith is an open device based on the ESP32-WROOM-32 module. This module consists of an ESP32 system on chip (SoC), with an RF section supporting 2.4 GHz Wi-Fi and Bluetooth 4.2, and 4 MB of serial flash memory. The ESP32 is a dual-core chip clocked at 240 MHz, with two Xtensa LX6 CPUs based on the Harvard architecture. Its internal memory comprises 448 KB of ROM and 520 KB of SRAM. The board includes a CP2102 or CH9102 USB-to-UART bridge.

The software development environment uses the Arduino Integrated Development Environment (IDE) and the Oracle Java Card Development Kit (JDK). Four Java Card applications (.cap files) are available: *TLS-SE.cap* (TLS for Secure Element), *CC.cap* (Crypto Currency), *TLS-IM.cap* (extended TLS Identity Module), and *TLS-IM0.cap* (minimal TLS Identity Module).

APP	APP	APP	APP	GPShell	Terminal	LeMonolith	
TLS PSK	APDU	CMD SHELL	TLS Client	APDU	CMD SHELL	USB	
				winscard.dll		winscard.dll	
RFCOMM			TCP/IP	SERIAL		RACS	TLS
Bluetooth			Wi-Fi	USB		TCP/IP (IoSE)	
LeMonolith							

LeMonolith development kit (LEM DevKit) is a set of software tools that perform the following operations:

- Loading software onto the ESP32 SoC and Java Card.
- Using the smartcard through the USB interface (as a smartcard reader).
- Using the smartcard through the TLS-SE (TLS for Secure Element) Wi-Fi interface.
- Using the smartcard as a TLS identity module (TLS-IM) for a Wi-Fi TLS server; a smartcard reader is available over Wi-Fi, secured by TLS-PSK (TLS with a pre-shared key).
- Using the smartcard through the Bluetooth interface, including an Android mobile application and TLS-SE services over Bluetooth.

3 Initialization of the LEM DevKit

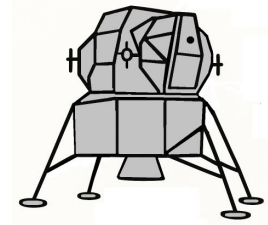
To install CP210x drivers for Windows, see

https://www.silabs.com/documents/public/software/CP210x_Windows_Drivers.zip

To install CH9102 drivers for Windows, see

https://www.wch-ic.com/downloads/CH343SER_ZIP.html

- Connect LeMonolith to a USB port.
- Go to /



In MAKE.bat, enter the IP address if you already know it:

set MYIP=192.168.1.35

To use the IoSE (*Internet of Secure Element*) server, comment out this line:

REM set MYIP=192.168.1.35

Run MAKE.bat to detect the USB serial port to which LeMonolith is connected.

Some LEM tools are compiled with Visual Studio 2005 (VS2005) and may require the Visual C++ 2005 Redistributable Package.

See <https://www.microsoft.com/en-us/download/details.aspx?id=26347>

The open-source GPSHELL software is compiled with Visual Studio 2015 (VS2015) and may require the Visual C++ 2015 Redistributable Package.

About GPSHELL <https://kaoh.github.io/globalplatform/>

The LEM directory /vcredist contains redistributable packages for 32-bit and 64-bit versions of Windows.

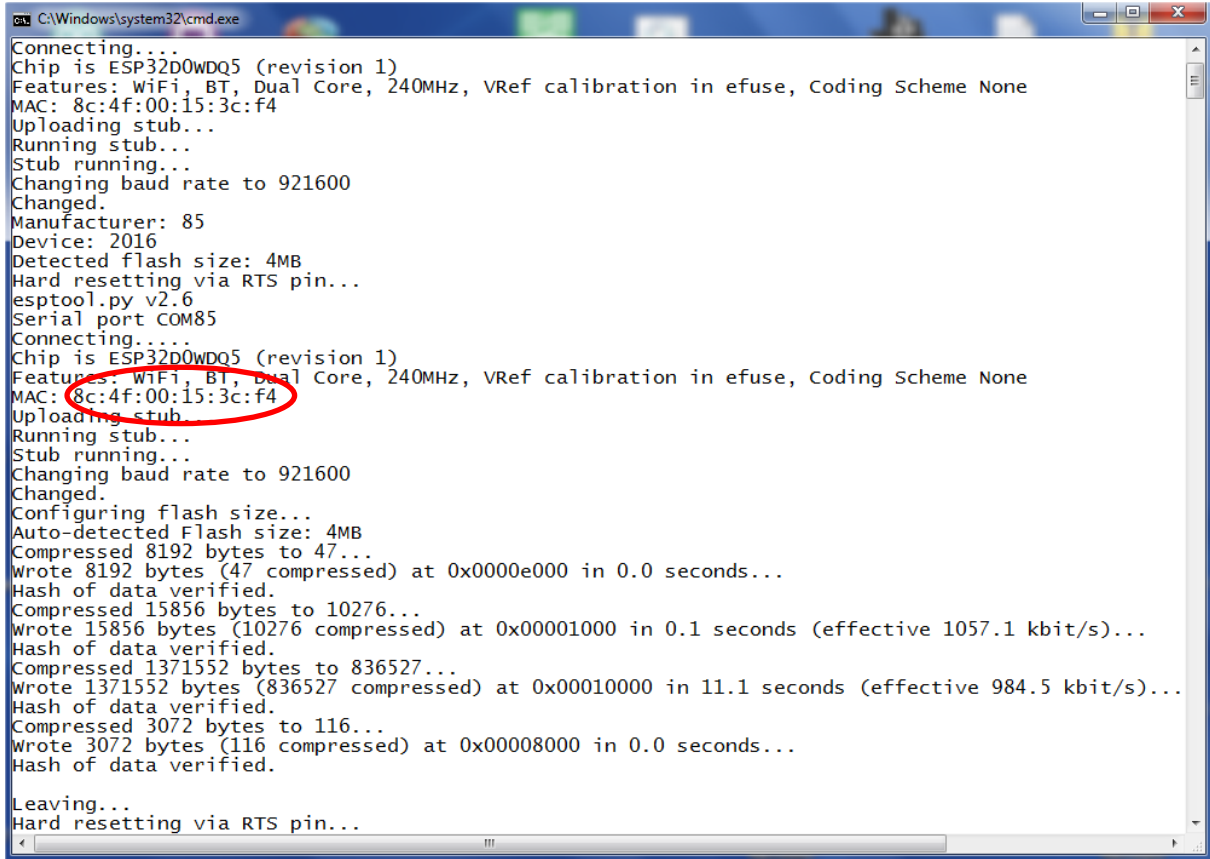
About Microsoft Visual C++ Redistributable

see <https://learn.microsoft.com/fr-fr/cpp/Windows/latest-supported-vc-redist?view=msvc-170>

4 Loading software

4.1 ESP32 software loader

Go to /ESP32loader and run loader.bat to load the latest version.



```
C:\Windows\system32\cmd.exe
Connecting....
Chip is ESP32D0WDQ5 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC: 8c:4f:00:15:3c:f4
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Manufacturer: 85
Device: 2016
Detected flash size: 4MB
Hard resetting via RTS pin...
esptool.py v2.6
Serial port COM85
Connecting....
Chip is ESP32D0WDQ5 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC: 8c:4f:00:15:3c:f4
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 8192 bytes to 47...
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.0 seconds...
Hash of data verified.
Compressed 15856 bytes to 10276...
Wrote 15856 bytes (10276 compressed) at 0x00001000 in 0.1 seconds (effective 1057.1 kbit/s)...
Hash of data verified.
Compressed 1371552 bytes to 836527...
Wrote 1371552 bytes (836527 compressed) at 0x00010000 in 11.1 seconds (effective 984.5 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 116...
Wrote 3072 bytes (116 compressed) at 0x00008000 in 0.0 seconds...
Hash of data verified.
Leaving...
Hard resetting via RTS pin...
```

In this example, the Wi-Fi MAC address is 8C:4F:00:15:3C:F4. The Bluetooth address (8C:4F:00:15:3C:F6) is obtained by adding 2 to the Wi-Fi address ($F4 + 2 = F6$). Once the software has been loaded, LeMonolith is in USB mode.

4.2 Factory operations

To load the ESP32 and secure element software, run:

- FACTORY_All.bat (use FACTORY_GP.bat)

To load only the secure element software, run:

- FACTORY_GP.bat install (im11c.cap, tls_se_305.cap, cc.cap, im0.cap)

4.3 Secure Element Software Loader

Go to /

4.3.1 Load applications into the secure element

- Run USB_GP_delete.bat to remove **all** Java Card applications.
- Run USB_GP_install.bat (or USB_GP_install2) to install all applications.

```
C:\Windows\system32\cmd.exe
mode_211
establish_context
card_connect
* reader name COM85/key
select -AID A000000151000000
open_sc -security 3 -keyind 0 -keyver 0 -mac_key 404142434445464748494a4b4c4d4e4f -enc_e4f
install -file im.cap
file name im.cap
install -file tls_se_guest.cap -priv 4
file name tls_se_guest.cap
card_disconnect
card_connect
* reader name COM85/key
send_apdu -sc 0 -APDU 00A4040006010203040500
send_APDU() returns 0x80209000 (Success)
card_disconnect
card_connect
* reader name COM85/key
send_apdu -sc 0 -APDU 00A4040006010203040500
send_APDU() returns 0x80209000 (Success)
card_disconnect
card_connect
* reader name COM85/key
select -AID A000000151000000
open_sc -security 3 -keyind 0 -keyver 0 -mac_key 404142434445464748494a4b4c4d4e4f -enc_e4f
install -file cc.cap
file name cc.cap
install -file im0.cap
file name im0.cap
card_disconnect
release_context
press ENTER for setting TLS-IM keys
```

4.3.2 List applications stored in the secure element

Run USB_GP_list.bat to list all Java Card applications.

Applications stored in the secure element are identified by a sequence of bytes called an AID (Application Identifier): TLS-SE AID=010203040500, CC AID=010203040601, TLS-IM AID=01020304050700, and TLS-IM0 AID=01020304050800.

```
C:\Windows\system32\cmd.exe
mode_211
establish_context
card_connect
* reader name COM85/key
select -AID A000000151000000
open_sc -security 3 -keyind 0 -keyver 0 -mac_key 404142434445464748494a4b4c4d4e4f -enc_key 40414243444546474849e4f
get_status -element 40 -noStop
AID
| State | Privileges | Version | Linked Security Do
--
| -- | -- | -- | --
010203040700 TLS-IM | Selectable | | 0000 | a000000151000000
--
| -- | -- | -- | --
010203040500 TLS-SE | Selectable | | 0000 | a000000151000000
| | Default Selected / Card Reset | |
--
| -- | -- | -- | --
010203040601 CC | Selectable | | 0000 | a000000151000000
--
| -- | -- | -- | --
010203040800 TLS-IM0 | Selectable | | 0000 | a000000151000000
card_disconnect
release_context
```

5 Operating mode selection

LeMonolith has three main operating modes:

- USB, which functions as a smartcard reader.
- Wi-Fi, which provides a personal hardware security module (pHSM) over TCP/IP.
- Bluetooth, for smartphone applications.

There are two ways to select the operating mode:

- By using the two push buttons
- By using a serial terminal

5.1 Using push buttons



- Hold down the ACK button.
- Briefly press the RESET button.
- Release the ACK button when the blue LED starts blinking.
- Press the ACK button twice to display the current operating mode.
- Press the ACK button twice to select another operating mode.
- Press the RESET button to restart LeMonolith.

Mode	Blue LED	Red LED
USB	On	On
Bluetooth	On	Off
Wi-Fi	Off	On
Bluetooth USB	Blinking	Blinking
Bluetooth TLS-PSK	Blinking	Off
Wi-Fi + TLS-IM	Off	Blinking

5.2 Using Serial Terminal

The serial interface uses 115200 baud, 8 data bits, and no parity.

The LeMonolith SDK provides a dedicated terminal and an older version of Windows HyperTerminal.

Run `TERM_terminal.bat` for the dedicated terminal, or `TERM_hyperterminal.bat` for Microsoft HyperTerminal.

```

C:\Windows\system32\cmd.exe
#dtr 0
#rts 1
#wait 1
#rts 0
#wait 1000
#flush
#more
Enter quit to exit
[default=usb]
Press ENTER or Double Click

ap ssid password
clear (erase eeprom)
version
exit
mode
[1=usb 2=bt 3=wifi quiet=mode+64 verbose=mode+32 bt_serialonly=98 bt_tlspk=18 tlim=51]
[usb]

shell>

```

LeMonolith can also be powered through USB On-The-Go (OTG) on Android and used with the "Serial USB Terminal" application. Use 115200 baud, 8 data bits, no parity, CR LF line endings, and local echo.

See https://play.google.com/store/apps/details?id=de.kai_morich.serial_usb_terminal

- Hold down the ACK button.
- Briefly press the RESET button.
- Release the ACK button when the blue LED starts

The following lines are displayed:

```
[default=usb]
Press ENTER or Double Click
```

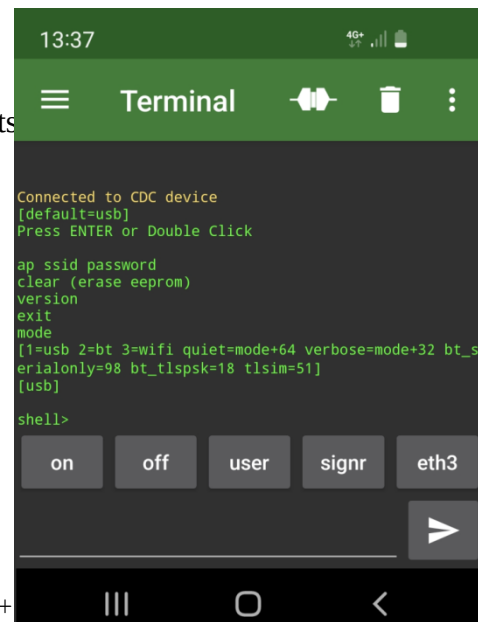
Press ENTER to open the configuration menu;
the following lines are displayed:

```

ap ssid password
clear (erase eeprom)
version
exit
mode
[1=usb 2=bt 3=wifi quiet=mode+64 verbose=mode+
bt_tlspk=18 tlim=51]
[usb]

shell>

```



To set the Wi-Fi SSID and password, enter the following command:
Type ap YourSSID YourPASSWORD and press ENTER.

```

>ap YourSSID YourPASSWORD
New ssid/passwd has been written in EEPROM
>shell

```

To select a mode, type *mode number* and press ENTER.

```

>mode 98
new mode 98 has been written in EEPROM
>shell

```

Mode	Comment
1	USB mode, works with TLS-SE shell (no debug)
2	BLUETOOTH mode, works with CC-SE (Crypto Currency) shell (debug)
3	Wi-Fi mode, works with TLS-SE (debug)
98	USB Bluetooth mode, works with CC-SE shell
18	Bluetooth with TLS-PSK (experimental)
33	USB debug mode
65	USB nodebug mode
34	Bluetooth debug mode
66	Bluetooth nodebug mode
35	Wi-Fi debug mode
67	Wi-Fi nodebug mode
51	Wi-Fi + TLS-IM debug mode
83	Wi-Fi + TLS-IM nodebug mode

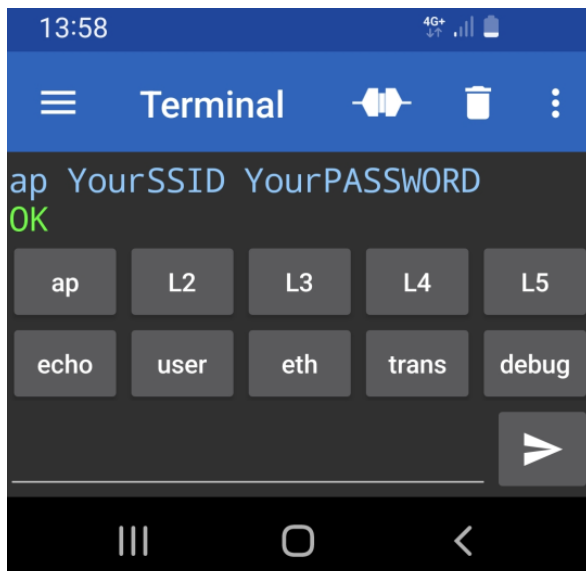
5.3 Setting Wi-Fi parameters in Bluetooth mode

LeMonolith is compatible with Android Bluetooth applications such as "Serial Bluetooth Terminal" (9600 baud, CR LF line endings, and local echo).

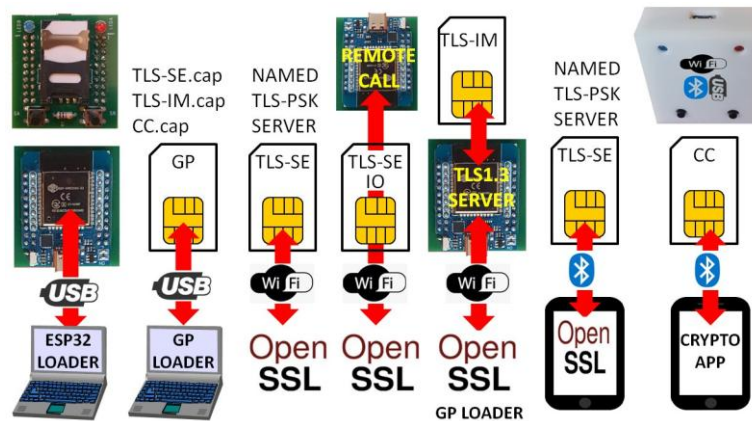
https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal

In Bluetooth mode, the ap command sets the SSID and PASSWORD parameters.

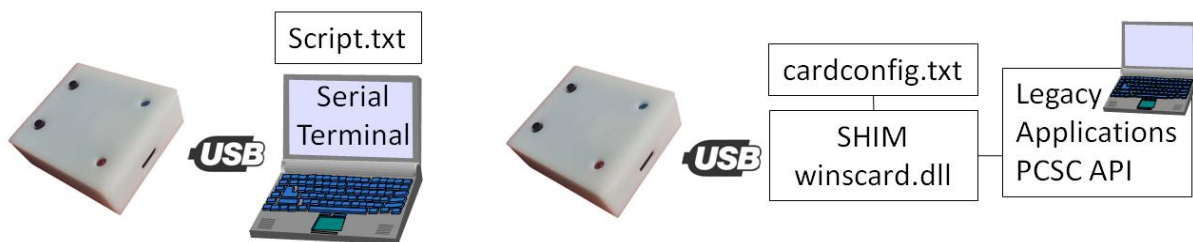
- Select Bluetooth mode.
- Pair with the *LeMonolith* device (RFCOMM profile, 9600 baud, no parity).
- Type ap YourSSID YourPASSWORD and press ENTER.



6 Overview



6.1 USB Mode



After the software is loaded, LeMonolith operates in USB mode. A command shell provides access to the secure element through the USB serial interface (115200 baud, no parity, one stop bit). It provides three main commands:

- **on** powers on the secure element
- **A** [ASCII hexadecimal-encoded APDU] sends an ISO7816 command to the secure element and returns the response.
- **off** powers off the secure element

Advanced commands (see section 7) can modify smartcard operating parameters such as frequency (**F**), baud rate (**pts**), and transmission protocol (**t0**, **t1**).

There are two ways to interface with the secure element:

- A serial terminal used with a script.
- A SHIM (*wincard.dll*, configured through *cardconfig.txt*) that bridges the PC/SC API and the USB serial interface. Open-source software such as GPSHELL (see <https://github.com/kaoh/globalplatform/tree/master/gpshell>) is compatible with LeMonolith when this SHIM is located in the software's directory.

#dtr 0	F
#rts 1	ifs
#wait 1	on
#rts 0	hist
#wait 1000	A 00A4040006010203040500
#flush	A 002000010830303030303030
#timeout 10000	A 0081000000
null	A 0089000000
null	A 0082000000
pts	test
ta	off

Example script for the serial terminal (config/USB00.bat)

6.2 USB Bluetooth mode

This mode is similar to USB mode but uses a different command shell (see section 8), dedicated to the CC (*CryptoCurrency*) Java Card application.

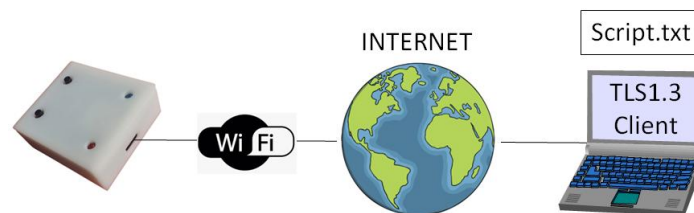
The script below (`./config/USB_TRANS.bat`) creates an Ethereum account (`genkey 2`) using the key at index 2 and generates an Ethereum transaction (`settransf`) in `trans.txt` (`#write`). The transaction **MUST** be confirmed by pressing the ACK button while the red LED is blinking.

#dtr 0	adm 00000000
#rts 1	genkey 2
#wait 1	eth 2
#rts 0	eip155 11155111
#wait 1500	#timeout 10000
#flush	settransf 2 45 10 100000
#timeout 2000	86F9E3E33BA7E42AB1128DA9291F675FA82546FF 0.0 #hello
#file trans.txt	#write
null	#timeout 2000
null	off

Diagram annotations: Red arrows point from labels to script components. 'key' points to 'eth 2'. 'nonce' points to '11155111'. 'GasPrice' points to '45'. 'GasLimit' points to '100000'. 'recipient address' points to the long hexadecimal string. 'amount data' points to '0.0'.

Serial terminal script for an Ethereum transaction (`./config/USB_TRANS.bat`)

6.3 Wi-Fi Mode



In this mode, LeMonolith is an Internet server that provides a command shell over TLS 1.3, allowing access from anywhere in the world. A TLS client such as OpenSSL, WolfSSL, or EtherSSL is required on the client side. Commands implemented by the TLS-SE Java Card application are listed in section 9.

Commands such as `c02` (clear keys at index 2), `g02` (generate keys at index 2), `p02` (get the public key at index 2), and `s02[data]` (sign data with the private key at index 2) are executed by the secure element over an end-to-end TLS session.

6.3.1 OPENSSL

Use the following command line to connect to LeMonolith with OpenSSL:

```
openssl s_client -tls1_3 -connect ip:444 -servername key1.com -groups P-256 -cipher DHE
-ciphersuites TLS_AES_128_CCM_SHA256 -debug -tlsextdebug -msg -no_ticket -psk
[PSK in hexadecimal]
```

6.3.2 WolfSSL

Commands can be sent using a dedicated TLS 1.3 client (`client.exe`) based on WolfSSL, for example:

- `client.exe -H #p02 -H #?02 -S key1.com -s -p 444 -h [LeMonolith-IP] -l TLS13-AES128-CCM-SHA256 -H psk[PSK in hexadecimal]`

The `parser2.exe` tool can extract responses recorded in a file, for example:

- `client.exe -H #p02 -H #?02 -S key1.com -s -p 444 -h [LeMonolith-IP] -l TLS13-AES128-CCM-SHA256 -H psk[PSK in hexadecimal] 1>log.txt`

The command `parse2.exe log.txt` creates `rx_i.txt` and `tx_i.txt` files containing the i^{th} request (`-H #request`) and i^{th} corresponding response.

6.3.3 EtherSSL

The `tlse.exe` software accepts the same arguments as `client.exe`, with an additional `-c` prefix. It supports multiple TLS identity modules hosted in smartcards, serial tokens, remote network readers, and TLSSE servers.

Example `tlse.exe` command line

`tlssset.exe -c -H #?00 -H #?02 -S key1.com -s -p 444 -h [LeMonolith-IP] -l TLS13-AES128-CCM-SHA256 -H psk[PSK in hexadecimal]`

6.3.4 Command list for the TLSSE client

option	comment	client.exe	tlse.exe
<code>-c, c</code>	Start a TLS-PSK client.	no	yes
<code>client</code>	Start a TLS-PSK client with default options.	no	yes
<code>tclient</code>	TLS-PSK test client with a TLSIM card	no	yes
<code>im</code>	TLS-PSK test server (PSK and PKI) with TLSIM. Wait for a message, then send a banner.	no	yes
<code>se</code>	TLS-PSK test server with a TLSSE card	no	yes
<code>sse</code>	TLS-PSK echo test server with a test TLSSE card (GitHub version)	no	yes
<code>-s</code>	Use PSK	yes, mandatory	yes not used
<code>-h [hostname]</code>	Server Name	yes	yes
<code>-p [port]</code>	Server Port	yes	yes
<code>-l [cipher suite]</code>	Server cipher suite TLS_AES_128_GCM_SHA256 TLS_AES_128_CCM_SHA256	yes	yes aesgcm aesccm
<code>-H</code>	Extended Option name[parameter]	yes	yes
<code>-H S[ServerName]</code>	Server name	yes	yes
<code>-H psk[PSKVALUE]</code>	Server PSK Key	yes	yes
<code>-H identity[IDVALUE]</code>	Server TLS-PSK Identity	yes	yes
<code>-H #[COMMAND]</code>	Send a command to the server; ! exits.	yes	yes
<code>-H *[COMMAND]</code>	Send a command to the server; ! exits.	no	yes
<code>-H @xyx[COMMAND]</code>	Send a command to the server xyz times	yes	yes
<code>-H ttcrypto</code>	Crypto tests	no	yes

-H ttlsim	TLSIM tests	no	yes
-H console	Open a console with the server	yes	yes
-H noconsole	No console with the server	no	yes
-H banner	Read a banner from the server	yes	yes
-H ![delay]	Wait delay ms after reading the banner.	yes	no
-H forever	Send a fixed message in console mode	yes, Hello World	yes, ?00
-H quiet	No debug messages	yes	yes
-H verbose	Verbose mode (default)	no	yes
-H timeout[delay]	Network response timeout in ms	no	yes
-H auth[PUBLICKEY]	Authenticate with CA public key	no	yes
-H sign [PRIVKEY]	Sign with CA private key	no	yes
-H hw[number]	Use a TLS identity module. 0=pcsc, 1=serial, 2=lemonolith, 101=MFA, 144=cryptoterminal	yes, only 0,1	yes, all
-H im	Use a TLS identity module through PC/SC.	yes	yes
-H ct	Use CryptoTerminal, with hw=1	yes	no
-H tc	Use MFA, with hw=1	yes	yes
-H mc	Use LeMonolith	no	yes
-H noim	No Identity Module	yes	yes
-H com[number]	Serial port number for Windows	yes	yes
-H serial[name]	Serial device name (/dev/tty...) for Linux	no	yes
-H baud[baud rate]	Baud rate for a serial TLSIM device	yes	yes
-H reset	Reset a serial TLSIM device using DTR or TLS.	no	yes
-H noreset	Do not reset the serial TLSIM device.		
-H rstwait[waitime]	Wait waitime ms after reset.	no	yes
-H			
-H aid[AIDVALUE]	AID parameter for tlsim	yes	yes
-H pin[PINVALUE]	PIN parameter for tlsim	yes	yes
-H rh[hostname]	remote server name	no	yes
-H rp[port]	remote server TCP port	no	yes
-H rl[CIPHERSUITE]	Remote server cipher suite TLS_AES_128_GCM_SHA256 - aesccm TLS_AES_128_CCM_SHA256 - aesgcm	no	yes
-H rS[ServerName]	remote server name	no	yes
-H rpsk[PSKVALUE]	remote server PSK	no	yes
-H ridentity[IDVALUE]	remote server identity	no	yes
-H rhw[number]	remote server hardware	no	yes
-H rim	Use PCSC tlsim for remote server	no	yes
-H rnoim	No identity module for remote server	no	yes
-H rimtlsse	remote server with binder/derive over TLSSE	no	yes
-H rimask	Remote server with binder/derive over TLSSE and button confirmation	no	yes
-H raid{AIDVALUE}	tlsim AID for remote server	no	yes
-H rpin[PINVALUE]	tlsim PIN for remote server	no	yes
-H rauth[PUBLICKEY]	Authenticate remote server	no	yes

-R rsign[PRIVATEKEY]	Sign remote server	no	yes
-H fkey[number]	Set the key index for encryption and decryption operations.	yes	yes
-H meta[metadata]	Set metadata for encryption and decryption operations.	yes	yes
-H Enc[filename] -H enc[filename]	Encrypt filename to filename.bin Enc: use hmac256 as key generator. enc: use BIP32 as key generator	yes	yes
-H Dec[filename] -h enc[filename]	Decrypt filename.bin to filename Dec: hmac256 as key generator dec: BIP32 as key generator	yes	yes

6.3.5 Using multiple identities

By default, the TLS-PSK identity is *Client_identity*, which is associated with the root account in the secure element.

The ?A501 command enables the guest account. The ?A500 command disables the guest account. The guest account is enabled by default. Its main commands read a memory record (Ixy, see section 10) and compute HMAC256 (hxy, see section 6.3.2).

Since TLS_SE version 3.03, the root account has managed the guest account's security policy.

Use the following command line to connect to a LeMonolith *psk-identity* account with OpenSSL:

```
openssl s_client -tls1_3 -connect ip:444 -servername key1.com -groups P-256 -cipher DHE
-ciphersuites TLS_AES_128_CCM_SHA256 -no_ticket -psk [PSK in hexadecimal]
-psk_identity [psk-identity]
```

Commands can be sent to a psk-identity account using a dedicated TLS 1.3 client (*client.exe*), for example:

```
client.exe -H #p02 -H #?02 -S key1.com -s -p 444 -h [LeMonolith-IP]
-l TLS13-AES128-CCM-SHA256 -H psk[PSK in hexadecimal] -H identity[psk-identity]
```

6.3.6 File encryption

An example is available in /cipher

A dedicated command computes HMAC256(key, message). The key is identified by its index, encoded as two hexadecimal digits (xy):

- hxy[message], h indicates HMAC256

This command generates key material (up to 32 bytes) for file encryption.

File encryption with *client.exe* requires three parameters:

- H **fkey***KeyIndex*, key index in hexadecimal form (xy= 00,01,...,ff).
- H **meta***MetaData*: metadata in text form.
- H **Enc***FileName*: the file name; **Enc** selects the HMAC256 key generator.

For example with:

KeyIndex=0, MetaData=metadata, filename= afile.txt

KeyIndex=1, MetaData=metadata2, filename= afile2.txt

The command line is:

```
client.exe -H fkey0 -H metametadata -H Encafile.txt -H fkey1 -H metametadata2 -H
Encafile2.txt -H com68 -H pin0000 -H noim -H identityguest -H
psk841657156D18FF6375E06B967F7E59B45243A8C01179079DDF35348B4DBBE93A -S
key21.com -s -p 8888 -h 185.216.10.85 -l TLS13-AES128-CCM-SHA256
```

The file to encrypt (FileName) is renamed FileName.bin and consists of two parts:

- Metadata terminated by a null character.
- AESGCM-encrypted content.

with

- KeyMaterial = HMAC256(KeyIndex, message) = 12-byte IV + 16-byte AES key + 4 unused bytes
- Message = SHA256(FileName || MetaData_ended_by_null_byte)
- IV= first 12 bytes of KeyMaterial
- AES key = the 16 bytes following the IV
- AuthenticatedData= FileName || MetaData_ended_by_null_byte
- EncryptedData= file content

File decryption with client.exe requires two parameters:

- H **fkey***KeyIndex*, key index in hexadecimal form (xy= 00,01,...,ff)
- H **Dec***FileName*: the encrypted file name ending in .bin; **Dec** selects the HMAC256 key generator.

For example with:

- KeyIndex=0, FileName= afile.txt.bin

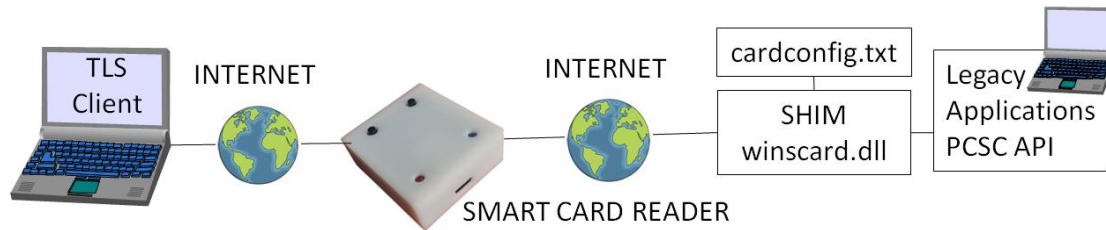
- KeyIndex=1, FileName= afile2.txt.bin

The command is:

```
client.exe -H fkey0 -H Decafile.txt.bin -H fkey1 -H Decafile2.txt.bin -H com68 -H pin0000
-H noim -H identityguest -S key21.com -s -p 8888 -h 185.216.10.85 -l TLS13-AES128-
CCM-SHA256 -H
psk841657156D18FF6375E06B967F7E59B45243A8C01179079DDF35348B4DBBE93A
```

The encrypted file (FileName.bin) is renamed FileName.

6.4 Wi-Fi TLS-IM Mode



In this mode, LeMonolith provides a TLS 1.3 server running on the ESP32 processor and secured by a TLS Identity Module (TLS-IM), the dedicated TLS-IM Java Card application. Two security mechanisms are supported:

- An X509 certificate, which supports only an echo operation;
- and a pre-shared key (PSK), which provides access to a command shell (see section 9).

The command `/TLSIM_LEM_Client_PKI_AESGCM_echo` provides an example using an X509 certificate with OpenSSL.

When TLS 1.3 PSK is used with the `TLS_AES_128_GCM_SHA256` cipher suite, *LeMonolith* functions as a remote smartcard reader (see `/TLSIM_LEM_Client_PSK_AESGCM_reader.bat`).

LeMonolith routes TLS packets to the TLS-SE Java Card application for the `TLS_AES_128_CCM_SHA256` cipher suite, and to the TLS-IM command shell for the `TLS_AES_128_GCM_SHA256` cipher suite.

Use the following command line to connect to LeMonolith with OpenSSL:

```
openssl s_client -tls1_3 -connect ip:444 -groups P-256 -cipher DHE -ciphersuites
TLS_AES_128_GCM_SHA256 -debug -tlsextdebug -msg -no_ticket
-psk [PSK in hexadecimaldecimal]
```

Commands can be sent using a dedicated TLS 1.3 client (*client.exe*), for example:

```
client.exe -H console -s -p 444 -h [LeMonolith-IP] -l TLS13-AES128-GCM-SHA256
-H psk[PSK in hexadecimaldecimal]
```

The *TLS-IM command shell* provides access to the secure element through three main commands:

- **on** powers on the secure element
- **A** [ASCII hexadecimal-encoded APDU] sends an ISO7816 request to the secure element and returns the response.
- **off** powers off the secure element

There are two ways to interface with the online smartcard reader:

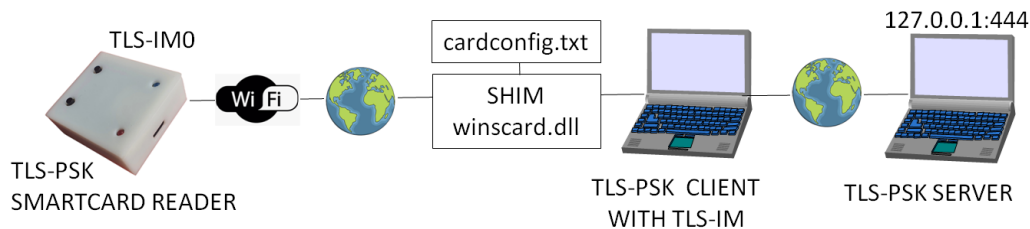
- a TLS1.3 client,
- A SHIM (*wincard.dll*, configured through *cardconfig.txt*) that bridges the PC/SC API and the USB serial interface.

Open-source software such as GPSHELL (<https://github.com/kaoh/globalplatform>) is compatible with LeMonolith when this SHIM is located in the software's directory.

```
seid 192.168.1.35:444/key
psk 0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
tx 1
```

Example cardconfig.txt file used by the wincard.dll SHIM

6.4.1 Example use of a remote smartcard reader



A TLS-PSK server is started on socket 127.0.0.1:444 with the command TLSIM_SERVER_LOCAL_PSK_AESCCM.bat.

```

C:\Windows\system32\cmd.exe
Using default temp DH parameters
ACCEPT
-----BEGIN SSL SESSION PARAMETERS-----
MHICAQECAGMEBAITBAQgwS1duLpyJkehww6OG9erNwN0E
IJHM5Ne3n1HfeVz6hE1/EAxnw0q3z4mcQ7dU3opV6A7/o
BgQEAQAAAKUDAgEBrgYCBG9jxp8=
-----END SSL SESSION PARAMETERS-----
Shared ciphers: TLS_AES_128_CCM_SHA256
Supported Elliptic Groups: P-384:P-256
Shared Elliptic groups: P-256
CIPHER is TLS_AES_128_CCM_SHA256
Reused session-id
Secure Renegotiation IS NOT supported
  
```

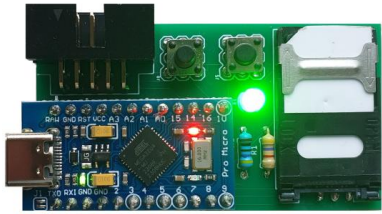
A TLS-PSK client, started with TLSIM_SERVER_LEM_CONNECT_REMOTE.bat, uses a TLS-IM0 smartcard through the PC/SC API. The wincard.dll SHIM enables remote access to this smartcard.

```

C:\Windows\system32\cmd.exe
peer has no cert!
SSL version is TLSv1.3
SSL cipher suite is TLS_AES_128_GCM_SHA256
SSL curve name is SECP256R1
Session timeout set to 500 seconds
Client Random : 73C01CE2F9BB62606AB4EC03CC8BDA7F97F89DA4A17907507B6581B4A5F0E03A
TLS1.3 Client is connected
Atr: 3B 07 52 41 43 53 30 30 31
Tx: 00 A4 04 00 06 01 02 03 04 08 00
Rx: 90 00 [185 ms]
Tx: 00 20 00 00 04 30 30 30 30
Rx: 90 00 [60 ms]
Tx: 00 85 00 0E 20 FB 8C 04 D1 D2 8A 0D CC C5 7A 5E
40 C0 98 F2 40 5B 49 02 92 7C FE 5B 85 97 42 02
39 81 0B FF 1D
Rx: D3 B8 7D 18 AF D4 1C A8 F8 48 5B 5E 65 3F 95 C7
A4 E4 F8 E2 D7 70 C0 87 39 3E A5 93 87 B1 CD 7A
90 00 [315 ms]
HS_Secret : D3B87D18AFD41CA8F8485B5E653F95C7A4E4F8E2D770C087393EA59387B1CD7A
key: D3B87D18AFD41CA8F8485B5E653F95C7A4E4F8E2D770C087393EA59387B1CD7A
TLS1.3 Client is connected in 9720 ms
Send (! or : to exit)
  
```

6.5 TLS Identity Module

TLS-IM tokens can be used on the client side to secure a TLS session. A TLS-IM token contains a smartcard with the TLS-IM Java Card application. Available form factors include smartcards and multi-factor authentication (MFA) modules.



Example of TLS-IM MFA Token

In *USB mode*, LeMonolith acts as a TLS-IM token. It can be accessed through the USB serial interface or through a *winscard.dll* SHIM for PC/SC.

The following `client.exe` command uses the USB serial interface (`com58`) to connect to a TLS 1.3 server at `127.0.0.1:444` with the server name `key1.com`:

- `client.exe -H $script.txt -H com58 -H hw1 -H aid010203040700 -H pin0000 -H console -S key1.com -s -p 444 -h 127.0.0.1 -l TLS13-AES128-CCM-SHA256`
- The file `script.txt` is used to start LeMonolith (for example `#dtr 0, #rts 1, #wait 1, #rts 0, #wait 1000, #flush`).

The following `client.exe` command uses PC/SC and the *winscard.dll* SHIM (see section 5.1) to connect to a TLS 1.3 server at `127.0.0.1:444` with the server name `key1.com`:

- `client.exe -H im -H aid010203040700 -H pin0000 -H console -S key1.com -s -p 444 -h 127.0.0.1 -l TLS13-AES128-CCM-SHA256`
- `tlsse.exe -c -H im -H aid010203040700 -H pin0000 -H console -S key1.com -p 444 -h 127.0.0.1 -l TLS13-AES128-CCM-SHA256`

6.6 Recursive Authentication with TLSSE client

LeMonolith hosts two TLS-PSK servers: one in the secure element, associated with the TLS13-AES128-CCM-SHA256 cipher suite, and one in the ESP32, associated with the TLS13-AES128-GCM-SHA256 cipher suite.

The TLSSE client connects to TLS-PSK servers using **-h** (host), **-p** (port), **-S** (server name), **-H psk** (pre-shared key), and **-l** (cipher suite). These options identify the server socket, name, pre-shared key, and cipher suite. Physical TLS identity modules (TLSIM), implemented in smartcards, MFA tokens, or LeMonolith devices, can prevent pre-shared-key hijacking. The **-H im** option selects an external TLSIM smartcard, and **-H hw(number)** selects a serial TLSIM device. A TLSIM module is identified by its application identifier (**-H aid(value)**) and protected by a PIN (**-H pin(value)**).

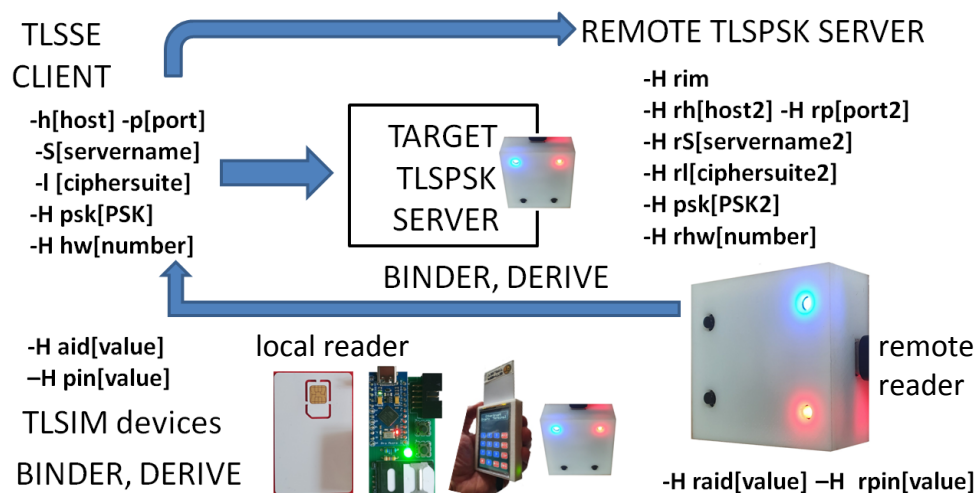
A remote TLS-PSK server can perform authentication procedures (*binder* and *derive*) for a connection to a target TLS-PSK server. This is a recursive process: a connection is first

established with the remote server, then its binder and derive functions are invoked to establish the session with the target server.

The **-H rim** option selects a remote server acting as a remote smartcard reader. Its attributes are specified by **-H rh(host)**, **-H rp(port)**, **-H rS(servername)**, **-H rpsk(pre-shared-key)**, and **-H rl(ciphersuite)**. The default cipher suite is TLS13-AES128-GCM-SHA256 (reader mode). The remote secure element's application identifier and PIN are specified by **-H aid(value)** and **-H pin(value)**. A physical TLS identity module (smartcard or serial device) can also be used to connect to the remote server; select it with **-H rim -H rhw(number)**. Its application identifier and PIN are specified by **-H raid(value)** and **-H rpin(value)**.

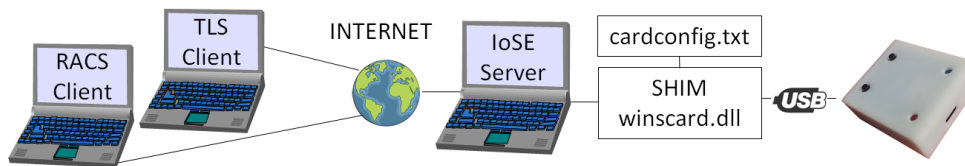
The **-H rmtlsse** option selects a remote TLSSE server. Its attributes are specified by **-H rh(host)**, **-H rp(port)**, **-H rS(servername)**, **-H rpsk(pre-shared-key)**, and **-H rl(ciphersuite)**. The default cipher suite is TLS13-AES128-CCM-SHA256 (TLSSE mode). A physical TLS identity module (smartcard or serial device) can also be used to connect to the remote server; select it with **-H rmtlsse -H rhw(number)**. Its application identifier and PIN are specified by **-H raid(value)** and **-H rpin(value)**.

The **-H rimask** option is similar to **rmtlsse** but asks the user to confirm the derive procedure by pressing the ACK button. A rapidly blinking blue LED indicates this request.



The `/tlsse` directory contains examples. Local TLS-PSK OpenSSL servers are started with `/tlsse/start_server_local_ccm.bat` (127.0.0.1:444, AESCCM) and `/tlsse/start_server_local_gcm.bat` (127.0.0.1:446, AESGCM).

6.7 IoSE server for Wi-Fi mode



The Internet of Secure Element (IoSE) server manages two TCP daemons: a *Remote APDU Call Secure* (RACS) server on port 7777 and a TLS 1.3 front-end server on port 8888. On Windows, the *wincard.dll* SHIM bridges PC/SC and the USB serial interface. The IoSE server identifies the secure element by its SEID (999) and secure element name, SEN (COMX001).

RACS uses TLS1.2 with X509 certificates for both server and client.

The LEM development kit provides tools to generate certificates for the certification authority (CA) in *makecert/MakeID/DemoC/makeca.bat*, and for the server (*CommonName=server*) and client (*CommonName=client*) in *makecert/MakeID/makeClientServer.bat*.

Use the following OpenSSL command line to connect to the RACS server:

- `openssl s_client -tls1_2 -connect 127.0.0.1:7777 -cipher AES128-GCM-SHA256 -cert client.pem -key clientkey.pem -CAfile root.pem -verify 1 -pass pass:pascal`

Use the following OpenSSL command line to connect to LeMonolith's embedded TLS 1.3 PSK server:

- `openssl s_client -tls1_3 -connect 127.0.0.1:8888 -servername COMX001 -groups P-256 -cipher DHE -ciphersuites TLS_AES_128_CCM_SHA256 -no_ticket -psk [PSK_Hexadecimal_Value]`

```

C:\Windows\system32\cmd.exe
r#0 COM34/key
Reader COM34/key is powered on... COMX001
ATR:3B 07 43 4F 4D 58 30 30 31
SEN= COMX001, AID= , for reader# 000
Reader(0) COM34/key is powerdown(sid=-1)...
Keystore server ready on port 127.0.0.1:8888 (total #sessions 0, #inuse 0, s=0)
RACS server ready on port 127.0.0.1:7777 (total #sessions 0)
  
```

Secure elements hosted by the RACS server are described in the *wincard.dll* SHIM configuration: *cardconfig.txt* specifies the server IP address (127.0.0.1), TCP port (7777), and SEID (999). The default CA file is *root.pem*, the default certificate is *client.pem*, and the default key file is *clientkey.pem*. This allows software such as GPHELL to manage secure element applications remotely and transparently.

seid 127.0.0.1:7777/999
tx 1

cardconfig.txt file for RACS SHIM

6.8 Bluetooth Mode



Bluetooth mode uses RFCOMM (*Radio Frequency Communication*), which emulates a serial port at 9600 baud. Two types of applications are available:

- Command shell (described in section 8), used from dedicated mobile applications.
- TLS-SE command shell (see section 10.1) used over TLS-PSK communications with the secure element.

7 LeMonolith Quick Testing

For these tests, LeMonolith must operate in Wi-Fi TLS-IM mode. The default PSK is used: `0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20`.

The IP address is stored in `myip.txt`.

All commands are located in the `./ssl` directory.

The `makecardconfig.bat` command creates the `cardconfig.txt` configuration file required by the `winscard.dll` SHIM.

```
seid 192.168.1.35:444/key
psk 0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
tx 1
```

Example of `cardconfig.txt` file

7.1 openssl.bat, WolfSSL.bat

These two commands open a TLS 1.3 session with the TLS-SE application using the AES_128_CCM_SHA256 cipher suite. In this scenario, the secure element acts as a TLS server.

7.2 openssl_AESGCM.bat, WolfSSL_AESGCM.bat

These two commands open a TLS 1.3 session with the LeMonolith server, secured by the TLS-IM application, using the AES_128_GCM_SHA256 cipher suite. In this scenario, LeMonolith acts as a smartcard reader over TLS.

7.3 openssl_PKI.bat

This command opens a TLS 1.3 session with the LeMonolith server, authenticated by a certificate and secured by the TLS-IM application. It uses the AES_128_GCM_SHA256 cipher suite and echoes the received TLS payload.

7.4 ListSE.bat

This command lists applications hosted by the secure element. It runs `gpshell.exe` with the `winscard.dll` SHIM.

7.5 DeleteSE.bat

This command deletes the TLS-SE and `cc.cap` Java Card applications from the secure element. It runs `gpshell.exe` with the `winscard.dll` SHIM.

7.6 LoadSE.bat

This command installs the TLS-SE (tls_se_305.cap) and cc.cap Java Card applications in the secure element. It runs gpshell.exe with the winscard.dll SHIM.

7.7 LoadSE2.bat

This command installs only the TLS-SE application tls_se_305.cap in the secure element. It runs gpshell.exe with the winscard.dll SHIM.

7.8 sign.bat

This command generates and delivers a certificate to the TLS-SE application.

7.9 auth.bat

This command establishes an authenticated TLS session with the TLS-SE application. It is part of the attestation procedure used before modifying the PSK.

7.10 TLS13_Server_PSK_AESCCM.bat

This command starts a local TLS server to demonstrate the TLS-IM security module.

7.11 connect_127_0_0_1_444_im_pcsc.bat

This command starts a TLS client secured by the TLS-IM module and demonstrates remote use of the TLS-IM application stored in LeMonolith.

7.12 reader.exe, winsreader.exe

These tools use PC/SC and the winscard.dll SHIM to provide transparent remote access to LeMonolith applications.

8 USB Command SHELL

The red LED is on when the smartcard is powered on.

The red LED blinks when a command is sent to the smartcard.

Command	Comments
test [number]*	Test n ECDSA signatures.
nodebug	nodebug mode
debug	Debug mode
F [frequency in KHz]	Get/Set the smartcard clock frequency (recommended value: 6000).
ta [value]	Get/Set the TA byte for the PTS protocol (recommended value: 12 in hexadecimal).
pts [value]	Get/Set the TA byte, TA = 0x10 + pts (recommended value: 2).
nopts	No PTS protocol (pts=0)
ptcol	Get the current T=x protocol (0=>T=0, 1=>T=1).
t0	Force T=0 protocol
t1	Force T=1 protocol
ifs [value]	Get/Set the IFS value for the T=1 protocol (recommended value: 254).
retry [number]*	Get/Set the retry count for the T=1 protocol (recommended value: 3)
finject [value]*	- bit 0 (1), inject CRC error for next T=1 request - bit 1 (2), inject CRC error for next T=1 response
on	Power on the smartcard.
off	Power off the smartcard.
hist	Get the historical bytes from the ATR.
A [APDU in hexadecimaldecimal]	Send an ISO7816 APDU in ASCII hexadecimal.

* not available for Wi-Fi TLS-IM mode

9 USB Bluetooth Command SHELL

The red LED is on when the smartcard is powered on.

Command	Comments
Empty	Return "ERROR No Command!"
echo	Return "OK"
nodebug	nodebug mode
debug	debug mode
F [frequency in KHz]	Get/Set the smartcard clock frequency (recommended value: 6000).
ta [value]	Get/Set the TA byte for the PTS protocol (recommended value: 12 in hexadecimal).
pts [value]	Get/Set the TA byte, TA = 0x10 + pts (recommended value: 2 or 3).
nopts	No PTS protocol (pts=0)
ptcol	Get the current T=x protocol (0=>T=0, 1=>T=1).
t0	Force T=0 protocol
t1	Force T=1 protocol
user PIN	Start the smartcard, select the CC-SE App, and supply the user PIN (four decimal digits, default: 0000).

changeuser oldpin newpin	Change the user PIN (4 decimal digits).
changeuser2 oldpin newpin	Change the user2 PIN (4 decimal digits).
changeadm oldpin newpin	Change the administrator PIN.
user2 PIN	Start the smartcard and supply the user2 PIN (for nonvolatile memory read/write operations only, default: 0000).
adm PIN	Start the smartcard, select the CC-SE App, and supply the user PIN (eight decimal digits, default: 00000000).
setlabel keyindex "text"	Associate a label with keyindex.
getlabel keyindex	Get the label for keyindex.
recover keyindex	~~Compute the recovery parameter (0 or 1) from a previous Ethereum transaction.~~
check	Check a signed CC-SE App with the EtherTrust public key
content	~~Return the transaction buffer~~
tecc	Elliptic curve library test
binder 32bytes	Compute cryptographic binder for TLS 1.3
derive 32bytes	Compute handshake secret for TLS 1.3.
sign keyindex value	Compute a canonical ECDSA signature for value (32 bytes).
signr keyindex value	Compute the canonical ECDSA signature and recovery parameter for value (32 bytes).
status	Read CC-SE App status
read adr size	Read size bytes (maximum 256) in nonvolatile memory at adr (decimal)
write adr hexavalue	Write bytes (in hexadecimal) at adr (decimal).
clear keyindex	Clear keyindex (1...15)
setseed keyindex hexavalue	Set BIP32 seed (up to 255 bytes in hexadecimaldecimal) for keyindex (1...15)
computekey keyindex path	Compute a key according to BIP32 using keyindex; path is a sequence of integers separated by '.' (i ₁ .i ₂ i _n).
setpp keyindex privk	Set the private and public keys at keyindex using the private key (privk).
setkey keyindex privk pubk	Set the public key (pubk) and private key (privk) at keyindex.
genkey keyindex	Generate a key at keyindex (1...15)
getpub keyindex	Read public key at keyindex (0,...,15)
getpriv keyindex	Read private key at keyindex (1,...,15)
getseed keyindex	Read BIP32 seed at keyindex
settransf param1= keyindex param2=Nonce (hexadecimal) param3=GasPrice in decimal GWEIs param4=GasLimit in decimal WEIs param5=Recipient Address (40 hexadecimal digits) param6=Amount in ETH floating point format(0.0) param7=Data #text or #\$hexadecimal	settransf 1 45 10 100000 86F9E3E33BA7E42AB1128DA9291F675FA82546FF 0.0 #hello settransf 1 45 10 100000 86F9E3E33BA7E42AB1128DA9291F675FA82546FF 0.0 \$1234 keyindex=1 nonce=45 GasPrice=10GWEI GasLimit=100000 amount=0.0 data=hello data=0x1234
btc keyindex [network ID]	BTC address associated with keyindex, with an optional network ID (0...255).

hash160 keyindex	BTC hash160 address associated with keyindex
eth keyindex	Ethereum address (20 bytes) associated with keyindex
eip155 decimal-value	Set EIP155 ChainID value (1= mainnet, 11155111=Sepolia)
ap SSID PASSWORD	Set the SSID and PASSWORD for Wi-Fi mode.

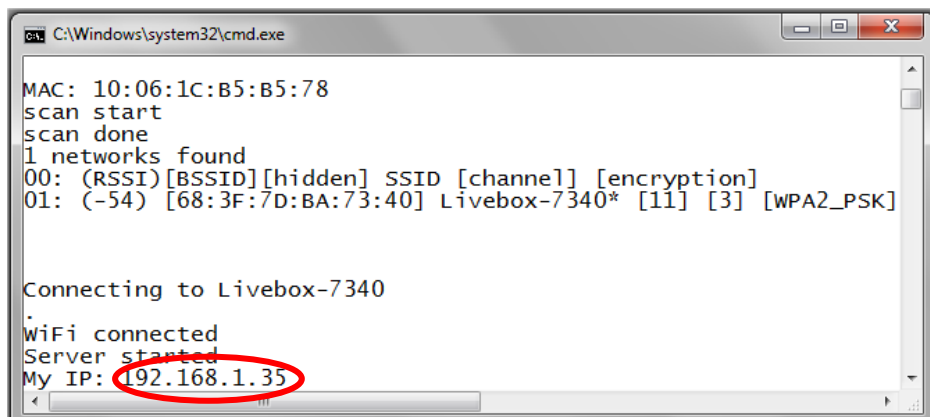
10 TLS-IM Wi-Fi Command Shell

The red LED is on when the smartcard is powered on.

The red LED blinks when a command is sent to the smartcard.

Command	Comments
nodebug	nodebug mode
debug	Debug mode
verbose 0/1	Verbose output disabled (0) or enabled (1).
F [frequency in KHz]	Get/Set the smartcard clock frequency (recommended value: 6000).
ta [value]	Get/Set TA byte for PTS protocol (recommended value is 12 in hexadecimal)
pts [value]	Get/Set the TA byte, TA = 0x10 + pts (recommended value: 2).
nopts	No PTS protocol (pts=0)
ptcol	Get the current T=x protocol (0=>T=0, 1=>T=1).
t0	Force T=0 protocol
t1	Force T=1 protocol
ifs [value]	Get/Set the IFS value for the T=1 protocol (recommended value: 254).
on	Power on the smartcard.
off	Power off the smartcard.
A [APDU in hexadecimal]	Send an ISO7816 APDU in ASCII hexadecimal.

11 Wi-Fi Operations



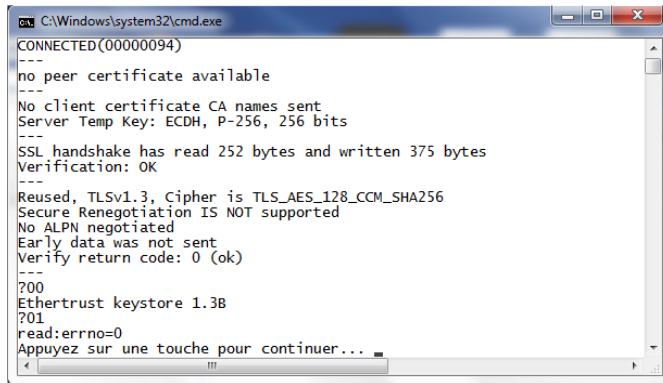
```
C:\Windows\system32\cmd.exe
MAC: 10:06:1C:B5:B5:78
scan start
scan done
1 networks found
00: (RSSI)[BSSID][hidden] SSID [channel] [encryption]
01: (-54) [68:3F:7D:BA:73:40] Livebox-7340* [11] [3] [WPA2_PSK]

Connecting to Livebox-7340
WiFi connected
Server started
My IP: 192.168.1.35
```

- The blue LED is on during a Wi-Fi scan.
- The blue LED is BLINKING when Wi-Fi is associated with an access point
- The blue LED is on while a TLS-PSK session is being established.
- The red LED is on when a TLS-PSK session is open.
- The red LED blinks during smartcard access.

11.1 Example OpenSSL command line

```
openssl s_client -tls1_3 -connect IP:444 -servername key1.com -groups P-256 -cipher DHE
-ciphersuites TLS_AES_128_CCM_SHA256 -no_ticket -psk
0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
```



When Wi-Fi TLS-IM is enabled, the following command line provides access to an online smartcard reader interface:

```
openssl s_client -tls1_3 -connect IP:444 -servername key1.com -groups P-256 -cipher DHE
-ciphersuites TLS_AES_128_GCM_SHA256 -no_ticket -psk
0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
```

12 TLS-SE Version 3.0.5

TLS-SE version 3.0.5 (TLS for Secure Element) is a Java Card 3.0.5 application designed for personal Network HSM (pNHSM) servers.

Access Control Policy	Key Management Service	Records
Command SHELL		
root	guest	
TLS1.3 ECDHE Pre-shared-keys Server		

psk_identity
pre-shared-key

Server Name
In ATR historical bytes
Up to 15 bytes

TLS-SE is a hardware security module accessed through shell commands over a protected TLS 1.3 connection.

Two accounts are available: root and guest. They use different PSK identities and pre-shared keys. The root account can enable or disable cryptographic features.

The embedded TLS 1.3 server uses pre-shared key (PSK) authentication, a mode resistant to quantum attacks, with an Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange over the SECP256 prime curve.

A TLS-SE app is associated with a server name stored in the ATR (Answer to Reset) historical bytes (up to 15 bytes).

A TLS-SE app has a public/private key pair (on the SECP256k1 curve) and a certificate. The Application Certification Procedure (ACP) verifies that the application is genuine.

TLS-SE provides two kinds of cryptographic services: KEYS associated with persistent cryptographic objects and SECRETS associated with on-demand cryptographic objects.

TLS-SE supports the following cryptographic procedures:

- RSA-2048 with PKCS #1 v1.5 signature.
- RSA-2048 encryption (in raw mode) and decryption.
- EC-256 (SECP256r1 and SECP256k1) with ECDSA signatures.
- AES-128 encryption and decryption.
- HMAC-SHA-256.
- HMAC-SHA-512.
- BIP32 with hardened keys.
- TLS 1.3 PSK binder and key derivation procedures.

Two wrap secrets are available, used to compute AES128 keys. Key blobs are built using AES-CCM authenticated encryption with associated data (AEAD) and a 16-byte tag.

TLS-SE provides four large files (up to 1,280 bytes) that can be used to store certificates and four small files (up to 64 bytes) that can be used to store messages.

12.1 Command Formats

A request contains a three-character ASCII header and a payload, followed by carriage-return (CR) and line-feed (LF) characters.

The payload format is text or binary.

Prefix: one character	Index: two hex digits	Payload (text or binary)	CR	LF
-----------------------	-----------------------	--------------------------	----	----

12.1.1 Request

In text format, CR may be omitted in a UNIX environment. In binary format, however, if the last payload byte is 0x0D, an additional CR character MUST precede LF.

A response contains a payload, followed by carriage-return (CR) and line-feed (LF) characters.

Payload (text or binary)	CR	LF
--------------------------	----	----

12.1.2 Response

A severe error is indicated by the TLS session closing; no response is returned.

12.2 Text Format

12.2.1 Request

Prefix: 1 character; index: 2 hexadecimal digits (0123456789ABCDEF); payload (up to 1,280 characters); CR LF

12.2.2 Response

Payload (up to 1,280 characters); CR LF
or
ERROR CR LF

12.3 Binary Format

12.3.1 Request

Prefix: 1 character; index: 2 hexadecimal digits (0123456789ABCDEF); payload (up to 1,280 bytes); CR LF

12.3.2 Response

Payload (up to 1,280 bytes); CR LF
or
Disconnection

12.4 Command Index Encoding

Commands are identified by a character prefix and divided into three logical groups: administrative (Admin), cryptographic (Crypto), and read/write (RW) commands.

Every command is associated with an index encoded as two hexadecimal digits.

12.4.1 Administrative Command

An Admin command uses the '?' prefix. It is identified by a two-hexadecimal-digit index representing an integer from 0 to 255.

12.4.2 Cryptographic Command

A cryptographic command has a character prefix. Its index, called KeyIndex, comprises a key identifier (KeyId) and an algorithm identifier (AlgoId).

KeyIndex is a byte denoted by $b_7b_6b_5b_4b_3b_2b_1b_0$. It is divided into three fields: Mode, AlgoId, and KeyId

b_7b_6 : Mode

b_7 : 1 = binary command; 0 = text command.

b_6 : 1 = multiple input; 0 = single input.

$b_5b_4b_3b_2$: AlgoId

0000 = EC256r1 or implicit; 0100 = EC256k1; 1000 = RSA2048; 1100 = Secret;

1100 = BIP32.

When the cryptographic procedure is identified by the command prefix, AlgoId is ignored and MAY be set to zero.

b_1b_0 : KeyId

Values range from 0 to 3.

12.4.3 Read/Write Command

For an RW command, the index is a record identifier (RecId) ranging from 0 to 7.

$b_2b_1b_0$: RecId

For records 0 to 3, the maximum size is 1,280 bytes.

For records 3 to 7, the maximum size is 64 bytes.

12.5 Blobs and Wrap Secrets

Two wrap secrets are available; both are used to compute AES-128 keys. Blobs are built using AES-CCM authenticated encryption with associated data (AEAD) and a 16-byte tag.

12.5.1 Blob Format

A blob stores an encrypted key or secret value. Its contents are divided into seven parts: BlobType, KeyIndex, KeyParam, BlobId, BlobRnd, EncryptedValue, and Tag, as shown below.

Blob: BlobType (1B) || KeyIndex (1B) || KeyParam (1B) || BlobId (4B) || BlobRnd (12B) || EncryptedValue || Tag (16B)

12.5.1.1 BlobType

BlobType identifies the blob version. It is set to zero or one for this version.

0= WrapSecret #0; WrapSecret #1.

12.5.1.2 KeyIndex

KeyIndex is the concatenation of AlgoId (4 bits) and KeyId (xy = 2 bits); the two most significant bits are set to zero.

00 0000 xy: SECP256k1

00 0100 xy: SECP256r1 (Prime)

00 1000 xy: RSA2048

00 1100 xy: Secret used for AES-128, HMAC-SHA-256, and HMAC-SHA-512

00 1110 xy: Secret seed used for BIP32

12.5.1.3 KeyParam

KeyParam identifies a subkey for a cryptographic algorithm.

For SECP256: 0 = public key; 0x80= private key.

For RSA2048: 0 = Modulus; 1 = PublicExponent; 2 = P; 3 = Q; 4 = DP1; 5 = DQ1; 6 = PQ.

For Secret: KeyParam = 0.

12.5.1.4 BlobRnd

BlobRnd is a 12-byte random number generated by the TLS-SE application.

12.5.1.5 BlobId

BlobId is a 4-byte value set by the user of the TLS-SE application.

12.5.1.6 EncryptedValue

EncryptedValue is the result of AES-CCM encryption of a key value.

The authenticated data (AD), a 19-byte field, is defined as follows:

AD = BlobType (1B) || KeyIndex (1B) || KeyParam (1B) || BlobId (4B) || BlobRnd (12B)

12.5.1.7 Tag

Tag is the 16-byte AES-CCM tag value.

12.5.2 Blob Computation

A wrap key is computed as follows:

BlobPrefix = "export01"

Message (24B) = BlobPrefix (8B) || BlobId (4B) || BlobRnd (12B)

WrapKey = HMAC-SHA-256(Secret, Message)

Secret: up to 64 bytes

The authenticated data (AD), a 19-byte field, is defined as follows:

AD = BlobType (1B) || KeyIndex (1B) || KeyParam (1B) || BlobId (4B) || BlobRnd (12B)

The key value is encrypted using AES-CCM with the WrapKey.

EncryptedKey = AES-CCM(WrapKey, KeyValue)

12.6 TLS-SE Components

12.6.1 Keys

Four RSA-2048 keys.

Four EC-256 keys for the SECP256r1 (prime) and SECP256k1 curves.

12.6.2 Secrets

Four secrets used for AES-128, HMAC-SHA-256, HMAC-SHA-512, and BIP32.

12.6.3 Wrap Secrets

Two wrap secrets.

12.6.4 Export/Import Keys

All keys and secret values can be exported or imported as cryptographic blobs built using AES-CCM authenticated encryption with associated data (AEAD) and a 16-byte tag.

12.6.5 Files

Four large files, up to 1,280 bytes.

Four small files, up to 64 bytes.

12.7 Performance

Performance is measured over an end-to-end TLS session using a local (127.0.0.1) IoSE server and a J3R180 Java Card:

- TLS 1.3 session establishment: < 1,000 ms
- TLS 1.3 1,000-byte echo: < 1,000 ms
- ECDSA signature: 128 ms (x8)
- RSA-2048 signature: 300 ms (x4)
- RSA-2048 encryption: 500 ms (x4)
- HMAC-SHA-256: 75 ms (x16)
- HMAC-SHA-512: 100 ms (x16)
- AES-128: 32 ms (x32)

12.8 TLS-SE App Commands, Version 3.05 (tls_se_305.cap)

Command	Description	G	Policy
?00	Version = 3.03	Y	5,01
?01[text]	Echo text	Y	5,02
?02	Disconnect (invalid command)	Y	always
?05[text]	Set root PSK identity	N	2,04
?06[text]	Set guest PSK identity	N	2,08
?0A	Get AppID (a SECP256k1 public key)	Y	2,10
?0B	Get app certificate: ECDSA (R, S) tuple, 64 bytes (128 hex digits)	Y	2,20
?0C[64 hex digits]	Authenticate (32-byte random number)	Y	2,40
?0E[128 hex digits]	Set certificate: ECDSA (R, S) tuple, 64 bytes (128 hex digits)	N	never
?30[128 hex digits]	Set secret for WrapSecret #0: up to 128 hex digits (64 bytes)	N	3,01
?31[128 hex digits]	Set secret for WrapSecret #1: up to 128 hex digits (64 bytes)	N	3,02
?33[14 hex digits]	Export key (14 hex digits, 7 bytes) BlobType (1B), KeyIndex (1B), KeyParam (1B), BlobId (4B) Return the blob value in hexadecimaldecimal, or ERROR	N	3,04
?34 [hex digits]	Import key BlobType (1B), KeyIndex (1B), KeyParam (1B), BlobId (4B) BlobRnd (12B), EncryptedValue, Tag (16B) Return OK or ERROR	N	3,08
?A0[64 hex digits]	Set PSK2 (used with the guest account); return the PSK2 value	N	4,01
?A1	Get PSK2 (used with the guest account)	N	4,04
?A2[64 hex digits]	Set PSK3 for the TLS-SE Binder and Derive procedures	N	4,02
?D0[64 hex digits]	Compute the Binder procedure using PSK3 and a 32-byte input value.	Y	4,10
?D1[64 hex digits]	Compute the Derive procedure using PSK3 and a 32-byte input value	Y	4,20
?D2[64 hex digits]	Compute the Derive procedure using PSK3 and a 32-byte input value. Request an ACK	Y	4,40
?A5[2 hex digits]	Bit 0 (0x01): enable the guest PSK identity Bit 7 (0x80): check the server name	N	never
?A8[2 hex digits] [2 hex digits]	Set access policy, First byte: block number; second byte: block value	N	never
?AA [OldPSK,NewPSK]	Set root PSK: OldPSK = 64 hex digits; NewPSK = 64 hex digits Return the NewPSK value	N	never
?FF[hex digits]	Echo a hexadecimal input value	Y	5,04
c[KeyIndex]	Clear Key KeyIndex= AlgoId KeyId	N	0,01
C[KeyIndex]	Clear Key KeyIndex= AlgoId KeyId, for AlgoId=0000 use SECP256P	N	0,01
g[KeyIndex]	Generate a key pair (SECP256P, SECP256k1, or RSA2048 only) KeyIndex= AlgoId KeyId	N	0,02
G[keyIndex]	Generate a key pair (SECP256P, SECP256k1, or RSA2048 only) KeyIndex= AlgoId KeyId, for AlgoId=0000 use SECP256P	N	0,02
*s[KeyIndex] [data]	Sign a value (up to 32 bytes); KeyIndex = Mode AlgoId KeyId SECP256P, SECP256k1, and RSA2048 only AlgoId may be 0000 for SECP256P and SECP256k1 Mode = b7b6; b7 = 0/1 (text/binary); b6 = 0/1 (single/multiple)	N	1,01
p[KeyIndex] [KeyParam]	Get public key; KeyIndex = AlgoId KeyId SECP256P, SECP256k1, and RSA2048 only. AlgoId may be 0000 for SECP256P and SECP256k1 For RSA2048, KeyParam is mandatory, 0 = Modulus; 1 = PublicExpo	N	0,04

r[KeyIndex] [Optional KeyParam]	Get private key; KeyIndex = AlgoId KeyId SECP256P, SECP256k1, and RSA2048 only AlgoId may be 0000 for SECP256P and SECP256k1 For RSA2048, KeyParam is mandatory: 2 = P; 3 = Q; 4 = DP1; 4 = DQ1; 5 = PQ	N	0,08
P[KeyIndex] [Optional KeyParam] [hex digits]	Set public key; KeyIndex = AlgoId KeyId SECP256P, SECP256k1, and RSA2048 only AlgoId may be 0000 for SECP256P and SECP256k1 (65B key) For RSA2048, KeyParam is mandatory: 0 = Modulus (256B); 1 = PublicExpo (4B)	N	0,10
R[KeyIndex] [Optional KeyParam] [hex digits] X	Set private key; KeyIndex = AlgoId KeyId SECP256P, SECP256k1, and RSA2048 only AlgoId may be 0000 for SECP256P and SECP256k1 (32B key) For RSA2048, KeyParam is mandatory: 2 = P; 3 = Q; 4 = DP1; 4 = DQ1; 5 = PQ	N	0,20
t[KeyIndex] [hex digits]	Set secret; KeyIndex = AlgoId KeyId Secrets only (AlgoId = 1100 or 1110) Secrets are used by BIP32, HMAC-SHA-256, HMAC-SHA-512, or AES procedures	N	1,02
T[KeyIndex] [hex digits]	Set secret; KeyIndex = AlgoId KeyId BIP32 secret only; up to 64 bytes (128 hex digits) AlgoId is ignored	N	1,02
b[KeyIndex] [Path hex digits] k	Compute a BIP32 tree; KeyIndex = AlgoId KeyId AlgoId is ignored Path is a sequence of 32-bit values with the MSB set (hardened key)	N	1,04
v[KeyIndex]	Get secret; KeyIndex = AlgoId KeyId AlgoId is ignored	N	1,08
*h[KeyIndex] [message]	Compute HMAC-SHA-256(secret, message) KeyIndex = Mode AlgoId KeyId Mode = b7b6; b7 = 0/1 (text/binary); b6 = 0/1 (single/multiple)	Y	0,80
*H[KeyIndex] [message]	Compute HMAC-SHA-512(secret, message) KeyIndex = Mode AlgoId KeyId Mode = b7b6; b7 = 0/1 (text/binary); b6 = 0/1 (single/multiple)	Y	0,80
m[KeyIndex] [message]	Compute HMAC-SHA-256(secret, message) KeyIndex = KeyId, Mode and AlgoId are ignored The message is in text format; the response is hexadecimal text.	Y	0,80
*Axy[data]	AES encryption; KeyIndex = Mode AlgoId KeyId AES128(Secret, data); secret MUST be 16 bytes long AlgoId is ignored Mode = b7b6; b7 = 0/1 (text/binary); b6 = 0/1 (single/multiple)	Y	0,40
*axy[data]	AES decryption; KeyIndex = Mode AlgoId KeyId AES128(Secret, data); secret MUST be 16 bytes long AlgoId is ignored Mode = b7b6; b7 = 0/1 (text/binary); b6 = 0/1 (single/multiple)	Y	0,40
*Exy[data]	RSA2048 raw private encryption; KeyIndex = Mode AlgoId KeyId, AlgoId is ignored Mode = b7b6; b7 = 0/1 (text/binary); b6 = 0/1 (single/multiple)	N	1,01
*Dxy[data]	RSA2048 raw private decryption KeyIndex = Mode AlgoId KeyId, AlgoId is ignored Mode = b7b6; b7 = 0/1 (text/binary); b6 = 0/1 (single/multiple)	N	1.01
*exy[data]	RSA2048 raw public encryption KeyIndex = Mode AlgoId KeyId, AlgoId is ignored Mode = b7b6; b7 = 0/1 (text/binary); b6 = 0/1 (single/multiple)	N	1.01

*dxy[data]	RSA2048 raw public decryption KeyIndex = Mode AlgoId KeyId, AlgoId is ignored Mode = b7b6; b7 = 0/1 (text/binary); b6 = 0/1 (single/multiple)	N	1,01
Z[RecIndex] [hex digits]	Write a value to record RecIndex 0...3: 1,280-byte maximum; 4...7: 64-byte maximum	N	1,10
Ixy	Read a value from record RecIndex	Y	1,20

* The s, h, H, A, a, E, D, e, d commands use *Special Operation Mode* (SOM); see the next section.

Policy - A policy element consists of a block number (one byte) and a value (one byte) that identify a command.

12.9 Special Operation Mode (som)

For the s, h, H, a, A, E, D, e, d commands, the KeyIndex parameter uses a dedicated format to support binary commands and/or multiple-input (pipeline) operations.

KeyIndex= Mode || AlgoId || KeyId

Mode= b₇b₆

- b₇ selects binary format instead of hexadecimal format.
A binary command is encoded with a three-character prefix, a binary payload, and a two-character suffix (CR, LF).
The response contains a binary payload and a two-character suffix (CR, LF).
In the event of a fatal error, the connection is closed and no data is returned.
An error MAY be indicated by an empty payload followed by a two-character suffix (CR, LF).
- b₆ indicates a multiple-input (pipeline) command. The payload is divided into 32-byte blocks for HMAC, 16-byte blocks for AES-128, 32-byte blocks for ECDSA and RSA-2048 signatures, and 256-byte blocks for RSA-2048 encryption/decryption. Each block is used as input to the corresponding operation.

In summary:

- KeyIndex = 00 || AlgoId || KeyId identifies a single command using a hexadecimal payload and key KeyId.
- KeyIndex = 10 || AlgoId || KeyId identifies a single command using a binary payload and key KeyId.
- KeyIndex = 01 || AlgoId || KeyId identifies a multiple-block command using a hexadecimal payload and key KeyId. HMAC uses 32-byte blocks, AES uses 16-byte blocks, and signatures use 32-byte blocks.
- KeyIndex = 11 || AlgoId || KeyId identifies a multiple-block command using a binary payload and key KeyId.

12.10 Access Policy

The root account has access to all commands.

For the guest account, access policy is controlled by six one-byte blocks. The root account can enable or disable individual commands using the ?A8[block#][value] command.

Block 0	b7	b6	b5	b4	b3	b2	b1	b0
cmd	h, H, m	a, A	R, X	P	r	p	g, G	c, C
Default: 0xC4	1	1	0	0	0	1	0	0

Block 1	b7	b6	b5	b4	b3	b2	b1	b0
cmd			I	Z	v	k, b	t, T	s, E,D,e,d
Default: 0x25	0	0	1	0	0	1	0	1

Block 2	b7	b6	b5	b4	b3	b2	b1	b0
cmd		?0C	?0B	?0A	?06	?05		
Default: 0x70	0	1	1	1	0	0	0	0

Block 3	b7	b6	b5	b4	b3	b2	b1	b0
cmd					?34	?33	?31	?30
Default: 0x00	0	0	0	0	0	0	0	0

Block 4	b7	b6	b5	b4	b3	b2	b1	b0
cmd		?D2	?D1	?D0		?A1	?A2	?A0
Default: 0x70		1	1	1	0	0	0	0

Block 5	b7	b6	b5	b4	b3	b2	b1	b0
cmd						?FF	?01	?00
Default: 0x07	0	0	0	0	0	1	1	1

12.11 TLS-SE Application Certification Procedure (ACP)

TLS-SE creates a public/private key pair upon instantiation. The *Application Certification Procedure* (ACP) reads the public key and writes a certificate (an ECDSA signature of the public key, consisting of two 32-byte values, R and S).

```
// GetID= Get Application Public Key (over elliptic curve Secp256k1)
?0A
043288117A7871F1CC92E3204D444BD9E656C2047D4FCE189F2F3F22AF01B07D2665F0C5332
06333E37454A8D00A2803E07BFF7356ED6AE74D94D874334A022AEF
// Set Certificate= ECDSACAPrivKey(SHA2(AppPubKey))= 64 bytes= R || S
?0E55D20B301E6E6A543B8FF2DA1F7C42371042A88A556CF4ECD0E76BF9740C51C5D0CF9741
2BA12B6A8640BA48A90D3B6CA18C87981D7E95E0B7D3FEDEE068D2CF
OK
```

12.12 TLS-SE Session Authentication Procedure (SAP)

The *Session Authentication Procedure* proves that the remote node knows the TLS-SE private key (TLS-SE-PrivKey) and the TLS handshake secret.

```
// GetID= Application Public Key
?0A
>>043288117A7871F1CC92E3204D444BD9E656C2047D4FCE189F2F3F22AF01B07D2665F0C53
3206333E37454A8D00A2803E07BFF7356ED6AE74D94D874334A022AEF
// Get TLS-SE Certificate
?0B
>>55D20B301E6E6A543B8FF2DA1F7C42371042A88A556CF4ECD0E76BF9740C51C5D0CF97412
BA12B6A8640BA48A90D3B6CA18C87981D7E95E0B7D3FEDEE068D2CF
// Authenticated Session Procedure
// Sign= Authenticate(32 bytes random)
// return ECDSATLS-SE-PrivKey(SHA2(HandshakeSecret || Random))
?0C7F69B857C6C2675BC8D5238E3E8BFC4C633FB5E39DD07F4760F508084FD1B482
>>304402206D2E688731C2673F977BD49B37D6CEC966323E966E34DE426D424AC5506F4A4B0
2203F5462E3D0AA7A1ED410ADDB29AE7C980EFC0136028FDF8533843D6A3C854ABF
```

12.13 TRAIT command

A trait command adds a '.' character at the beginning of the three-character prefix.
A trait command returns a response after execution and closes the TLS session.

A trait command can be used with OpenSSL to echo the hexadecimal response payload, as illustrated below:

```
echo -ne ".command\r\n" | openssl s_client -quiet other_parameters... 2>nul
```

SSL_openssl_trait.bat demonstrates this mechanism.

12.14 TLS-SE-IO commands

TLS-SE-IO is a way to export command from secure element and then to return response. A TLS-SE-IO command is identified by a '#' prefix.

Command	Comment
#on	Blue LED on
#off	Blue LED off
#on\$[decimal value]	LED on value=0=>blue, value=1=>red
#off\$[decimal value]	LED on value=0=>blue, value=1=>red
#ask	The blue LED blinks. If the push button is pressed within 5 seconds, the command returns OK; otherwise, it returns ERROR.
#read	Read voltage on GPIO34
#read2	Read voltage on GPIO35
#vbat	Read voltage on GPIO35, corrected value, 2,5*input
#charge	Battery state (Full, High, Low, Critical)

13 Bluetooth Operations

APP	APP	APP	APP	GPSShell	Terminal		LeMonolith
TLS PSK	APDU	CMD SHELL	TLS Client	APDU	CMD SHELL	USB	
				winscard.dll		winscard.dll	
RFCOMM			TCP/IP	SERIAL		RACS	TLS
Bluetooth			Wi-Fi	USB		TCP/IP (IoSE)	
LeMonolith							

Command	Comments
Empty	Return "ERROR No Command!"
echo	Return "OK"
user	Only for Bluetooth Serial, return OK
eth keyindex.PIN	Only for Bluetooth serial Ethereum address (20 bytes) associated with key index User PIN: 4 decimal digits
eip155 decimal-value	Set EIP155 ChainID value (1= mainnet, 11155111=Sepolia)
settransf param1= keyindex param2=Nonce (hexadecimal) param3=GasPrice in decimal GWEIs param4=GasLimit in decimal WEIs param5=Recipient Address (40 hexadecimal digits) param6=Amount in ETH floating point format(0.0) param7=Data #text or #\$hexadecimal	settransf 1 45 10 100000 86F9E3E33BA7E42AB1128DA9291F675FA82546FF 0.0 #hello settransf 1 45 10 100000 86F9E3E33BA7E42AB1128DA9291F675FA82546FF 0.0 \$1234 keyindex=1 nonce=45 GasPrice=10GWEI GasLimit=100000 amount=0.0 data=hello data=0x1234
Not available for Bluetooth Serial builds compiled with the btstrict option.	
nodebug	nodebug mode
debug	debug mode
pts [value]	Get/Set the TA byte, TA = 0x10 + pts (recommended value: 2).
nopts	No PTS protocol (pts=0)
ptcol	Get the current T=x protocol (0=>T=0, 1=>T=1).
t0	Force T=0 protocol
t1	Force T=1 protocol
on	Power on the smartcard and select CC-SE-APP.
on2	Power on the smartcard.
off	Power off the smartcard.
A [hexadecimal digits]	Only for USB debug. Send APDU
prompt	Prompt (>) is displayed
noprompt	Prompt (>) is not displayed
user PIN	For USB debugging only: start the smartcard, select the CC-SE App, and supply the user PIN (four decimal digits, default: 0000).

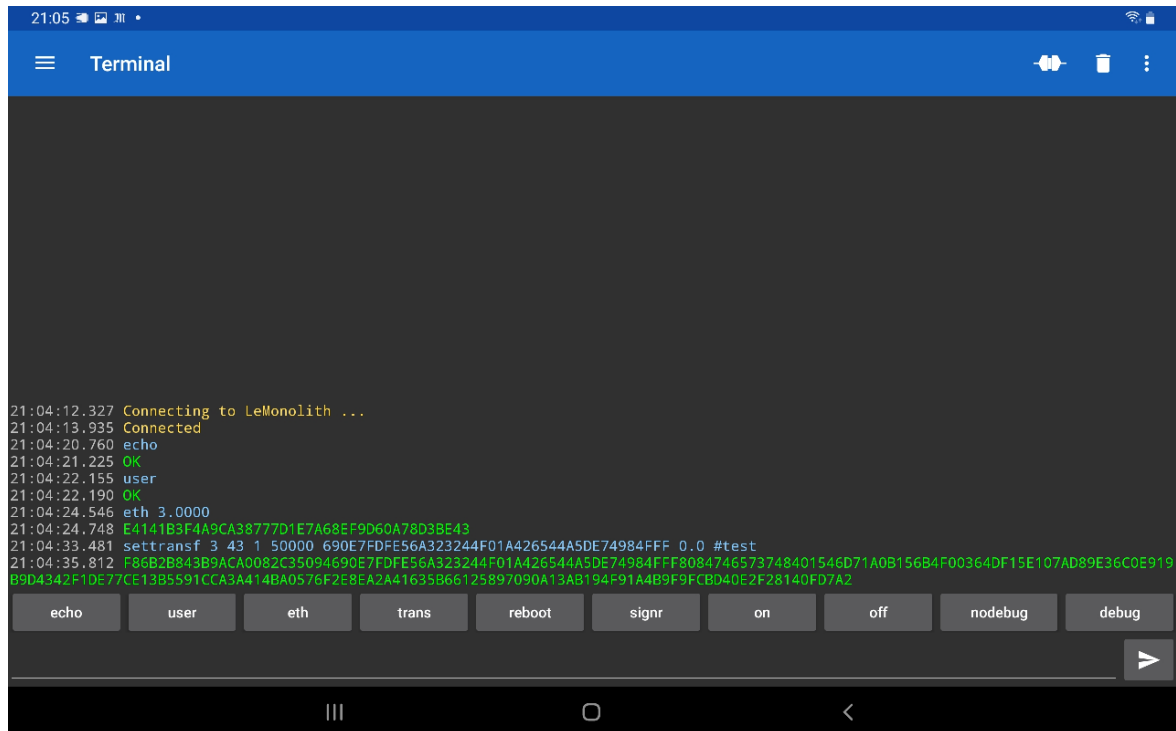
changeuser oldpin newpin	Change the user PIN (4 decimal digits).
changeuser2 oldpin newpin	Change the user2 PIN (4 decimal digits).
changeadm oldpin newpin	Change the administrator PIN.
user2 PIN	Start the smartcard and supply the user2 PIN (for nonvolatile memory read/write operations only, default: 0000).
adm PIN	Start the smartcard, select the CC-SE App, and supply the user PIN (eight decimal digits, default: 00000000).
setlabel keyindex "text"	Associate a label with keyindex.
getlabel keyindex	Get the label for keyindex.
recover keyindex	~~Compute the recovery parameter (0 or 1) from a previous Ethereum transaction.~~
check	Check a signed CC-SE App with the EtherTrust public key
content	~~Return the transaction buffer~~
tecc	Elliptic curve library test
~~binder 32bytes~~	~~Compute cryptographic binder for TLS 1.3~~
~~derive 32bytes~~	~~Compute handshake secret for TLS 1.3.~~
sign keyindex value	Compute a canonical ECDSA signature for value (32 bytes).
signr keyindex value	Compute the canonical ECDSA signature and recovery parameter for value (32 bytes).
status	Read CC App status
read adr size	Read size bytes (maximum 256) in nonvolatile memory at adr (decimal)
write adr hexavalue	Write bytes (in hexadecimal) at adr (decimal).
clear keyindex	Clear keyindex (1...15)
setseed keyindex hexavalue	Set BIP32 seed (up to 255 bytes in hexadecimaldecimal) for keyindex (1...15)
computekey keyindex path	Compute a key according to BIP32 using keyindex; path is a sequence of integers separated by '.' ($i_1.i_2.....i_n$).
setpp keyindex privk	Set the private and public keys at keyindex using the private key (privk).
setkey keyindex privk pubk	Set the public key (pubk) and private key (privk) at keyindex.
genkey keyindex	Generate a key at keyindex (1...15)
getpub keyindex	Read public key at keyindex (0,...,15)
getpriv keyindex	Read private key at keyindex (1,...,15)
getseed keyindex	Read BIP32 seed at keyindex
btc keyindex [network ID]	BTC address associated with keyindex, with an optional network ID (0...255).
hash160 keyindex	BTC hash160 address associated with keyindex

13.1 Serial Bluetooth terminal

The "Serial Bluetooth Terminal" application is compatible with LeMonolith.

See https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal

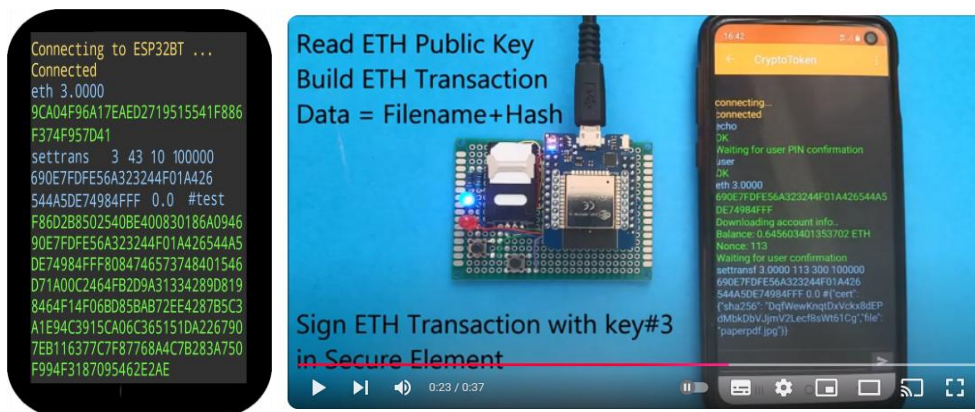
LeMonolith is compatible with the *Serial Bluetooth Terminal* application using 9600 baud, CR LF line endings, and local echo.



13.2 Bluetooth CryptoToken App for Android

The Android application is located in /Android/monolith.apk. Its functionality is similar to that of <https://play.google.com/store/apps/details?id=pascal.urien.cryptoterminal>.

This application creates Ethereum transactions with LeMonolith and signs a file stored on the smartphone on the Ethereum blockchain. See the demonstration video on YouTube: https://www.youtube.com/watch?v=O8b_yfAkqRM



14 Bluetooth TLS-PSK (BTPSK)

In this mode, the TLS-SE server runs over Bluetooth RFCOMM, with TLS packets transmitted transparently over Bluetooth.

- Blue LED blinking: waiting for Bluetooth connection
- Blue LED on: Bluetooth connection established
- Blue LED off: TLS-PSK connection established
- Red LED on: smartcard powered
- Red LED blinking: smartcard access

14.1 Testing Bluetooth TLS-PSK



A proxy application (proxy.bin) for LeMonolith bridges a TCP/IP socket and Bluetooth RFCOMM. It is available for *the LeMonolith* device.

- Red LED on: the proxy is running.
- Blue LED on: a Bluetooth connection to LeMonolith is established.

```

C:\Windows\system32\cmd.exe
>>> ??? [length 0005]
17 03 03 00 35
>>> TLS 1.3 [length 0001]
16
>>> TLS 1.3, Handshake [length 0024], Finished
14 00 00 20 b9 37 09 5c 65 52 1f 34 be 68 e
72 79 3c d6 c4 85 40 a0 0f 95 89 a9 d7 45 d
7c 15 10 17
write to 0x27ca80 [0x28b2a0] (64 bytes => 64 (0
0000 - 14 03 03 00 01 17 03 03 00 35 9f 94 0
0010 - a9 04 d9 a3 5f 18 1b 5f b1 44 20 3b 42 9
0020 - 3e 65 65 64 47 60 72 61 0c 63 28 90 84 6
0030 - fe 45 6f 19 d0 56 50 48 c9 a7 2c 46 98 d
...
no peer certificate available
...
No client certificate CA names sent
Server Temp Key: ECDH, P-256, 256 bits
...
SSL handshake has read 252 bytes and written 37
Verification: OK
...
Reused, TLSv1.3, Cipher is TLS_AES_128_GCM_SHA2
Secure Renegotiation IS NOT supported
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
...
700
>>> ??? [length 0005]
17 03 03 00 16
>>> TLS 1.3 [length 0001]
17
write to 0x27ca80 [0x28618b] (27 bytes => 27 (0
0000 - 17 03 03 00 16 f4 63 d4 d6 04 af a3 81 6
0010 - c4 6d 1d a1 4d 33 b7 c2 e9 02 d1
read from 0x27ca80 [0x28203b] (5 bytes => 5 (0x
0000 - 17 03 03 00 2b
<<< ??? [length 0005]
17 03 03 00 2b
read from 0x27ca80 [0x282040] (43 bytes => 43 (
0000 - 19 70 b5 fa 67 d6 cb 9a 3b 4a 10 7a 3b 9
0010 - dd 1d 6c 42 b4 1b 48 ab 5f 22 63 d4 f2 a
0020 - 46 99 67 d9 4e 04 61 63 f9 a5 9f
<<< TLS 1.3 [length 0001]
17
Ethereum keystore 1.38

C:\Windows\system32\cmd.exe
Server TLS13 Ready...
Connection from 127.0.0.1
Rx Thread Ready
311 bytes received on NetRecv
311 bytes sent on Serial
Rx Header (161)
166 bytes received on serial
Rx Header (23)
28 bytes received on serial
Rx Header (53)
58 bytes received on serial
252 bytes NetSend
6 bytes received on NetRecv
6 bytes sent on Serial
58 bytes received on NetRecv
58 bytes sent on Serial
27 bytes received on NetRecv
27 bytes sent on Serial
Rx Header (43)
48 bytes received on serial
48 bytes NetSend

COM3 - Term V1
File Edit Setup Control Window Help
Recv: 5
Recv: (53)58
RcNET_BT
17030300359f9403f77fa904d9a35f181b5fb144203b4299e2743e6565644760
72610c6328908467102afe456f19d0565048c9a72c4698debc59
Tx: 00d800033a17030300359f9403f77fa9
04d9a35f181b5fb144203b4299e2743e
656564476072610c6328908467102afe
456f19d0565048c9a72c4698debc59
TxT1: 4
00403PDF
RxT1: 4
004002D3
Rx[217ms]:
9001
TLS OPEN
Recv: 5
Recv: (22)27
RcNET_BT
1703030016f463d4d604afa381623fa1c46d1da14d33b7c2e902d1
Tx: 00d800031b1703030016f463d4d604af
A381623fa1c46d1da14d33b7c2e902d1
TxT1: 4
00002025
RxT1: 4
00003291
Rx[82ms]:
170303002b1970b5fa67d6cb9a3b4a10
7a3b984ee3dd1d6c42b41b48ab5f2263
d4f2a6747e469967d94e046163f9a59f
9000
TxNET_BT
170303002b1970b5fa67d6cb9a3b4a107a3b984ee3dd1d6c42b41b48ab5f2263
d4f2a6747e469967d94e046163f9a59f
Sent: 48
Recv: []
```

A video is available on YouTube: <https://www.youtube.com/watch?v=WyI4OxVTzHM>

15 LeMonolith (LEM) Dev Kit Tests

15.1 USB Operations

Select the USB mode.

APP	APP	APP	APP	GPShell	Terminal		LeMonolith
TLS PSK	APDU	CMD SHELL	TLS Client	APDU	CMD SHELL	USB	
				winscard.dll		winscard.dll	
RFCOMM			TCP/IP	SERIAL		RACS	TLS
Bluetooth			Wi-Fi	USB		TCP/IP (IoSE)	
LeMonolith							

15.1.1 COM_List.bat

List COM ports.

15.1.2 COM_Find.bat

Detect LeMonolith and write its COM port number to com.txt.

15.1.3 TERM_hypercentinal.bat

Start hyperterminal.

15.1.4 TERM_terminal.bat

Start terminal.

15.1.5 USB_GP_list.bat

List Java Card applications stored in the secure element.

15.1.6 USB_GP_delete.bat

Delete all Java Card applications (TLS-IM, TLS-SE, CC, TLS-IM0) stored in the secure element.

15.1.7 USB_GP_install.bat

Install tls_se_guest_hmac.cap (TLS-SE App), cc.cap (Crypto Currency App), im.cap (TLS-IM app), and im0.cap (TLS-IM0) in the secure element

15.1.8 USB_KEYSTORE_Genkey00.bat

Create a key (for the curve SECP256k1) at index 0 in TLS-SE App.

15.1.9 USB_SERVER_TLSSE.bat

Start a proxy TLS server at TCP port 8888.

This allows a TLS session with the secure element to be tested at the socket address 127.0.0.1:8888.

15.2 Wi-Fi Operations

Select the Wi-Fi mode.

APP	APP	APP	APP	GPShell	Terminal		LeMonolith
TLS PSK	APDU	CMD SHELL	TLS Client	APDU	CMD SHELL	USB	
				winscard.dll		winscard.dll	
RFCOMM			TCP/IP	SERIAL		RACS	TLS
Bluetooth			Wi-Fi	USB		TCP/IP (IoSE)	
LeMonolith							

15.2.1 SSL_openssl.bat

Open a TLS session with OPENSSL.

15.2.2 SSL_openssl_trait.bat

Demonstrate a TRAIT command, with openssl.

15.2.3 SSL_wolfssl.bat

Open a TLS session with WOLFSSL.

15.2.4 SSL_wolfssl_test.bat

Demonstrate special operation mode (SOM) for commands h, H, a, and A.

15.2.5 SSL_openssl_guest.bat

Open a TLS session with OpenSSL using the guest psk-identity. Enable the guest identity with the ?A501 command.

15.2.6 SSL_wolfssl_guest.bat

Open a TLS session with WolfSSL using the guest psk-identity. Enable the guest identity with the ?A501 command.

15.2.7 SSL_wolfssl_MFA.bat

Open a TLS session with the Multi Form Authentication (MFA) TLS-IM token.

15.2.8 SSL_wolfssl_PCSC.bat

Open a TLS session with the TLS-IM smartcard.

15.2.9 KEYSTORE_NET_Load_Key.bat

Load a private key from keystore\eth\mypp.txt at index 3, with PSK=keystore\eth\myspk.txt.

15.2.10 KEYSTORE_NET_Load_Key_SC.bat

Load a private key from \keystore\eth\mypp.txt at index 3, with PSK=keystore\eth\myspk.txt, using a TLS-IM smartcard.

15.2.11 KEYSTORE_NET_Load_Key_MFA.bat

Load a private key from \keystore\eth\mypp.txt at index 3, with PSK=keystore\eth\myspk.txt, using a TLS-IM MFA token.

15.2.12 KEYSTORE_NET_test_sign.bat

Perform ECDSA signatures using the key at index 3.

15.2.13 CIPHER_NET_init.bat

Initialize HMAC256 secret 02 with the text value 1234 (hexadecimal: 0x31323334).
Initialize HMAC256 secret 03 with the text value 5678 (hexadecimal: 0x35363738).

15.2.14 CIPHER_NET_encrypt.bat

Encrypt two files and their metadata using AESGCM128, with key material computed by HMAC256 using secrets at indices 02 and 03.

15.2.15 CIPHER_NET_decrypt.bat

Decrypt two files and their metadata using AESGCM128, with key material computed by HMAC256 using secrets at indices 02 and 03.

15.3 Wi-Fi Operations with TLS-IM

15.3.1 TLSIM_GP_USB_LOADER_IM.bat

Load TLS-IM software into the secure element **in USB mode**.

15.3.2 TLSIM_GP_USB_DELETE_IM.bat

Delete TLS-IM software from the secure element **in USB mode**.

15.3.3 TLSIM_GP_USB_PERSO_IM.bat

In USB mode, load a PSK into the secure element and an X509 certificate for the echo demonstration.

15.3.4 TLSIM_GP_USB_LOADER_IM0.bat

Load TLS-IM0 software into the secure element **in USB mode**.

15.3.5 TLSIM_GP_USB_DELETE_IM0.bat

Delete TLS-IM0 software from the secure element **in USB mode**.

15.3.6 TLSIM_GP_USB_PERSO_IM0.bat

Load a PSK into the IM0 application **in USB mode**.

15.3.7 TLS_IM_NET_GP_LIST.bat

This command lists applications hosted by the secure element. It runs gpshell.exe with the winscard.dll SHIM.

15.3.8 TLS_IM_NET_GP_DELETE.bat

This command deletes the TLS-SE and cc.cap Java Card applications from the secure element. It runs gpshell.exe with the winscard.dll SHIM.

15.3.9 TLS_IM_NET_GP_LOAD.bat

This command installs the TLS-SE (tls_se_305.cap) and cc.cap Java Card applications in the secure element. It runs gpshell.exe with the winscard.dll SHIM.

15.3.10 TLS_IM_NET_GP_LOAD2.bat

This command installs only the TLS-SE application tls_se_305.cap in the secure element. It runs gpshell.exe with the winscard.dll SHIM.

15.3.11 TLSIM_LEM_Client_PSK_AESGCM_reader.bat

Connect to LeMonolith in Wi-Fi TLS-IM mode using the AESGCM cipher suite and a pre-shared key (PSK). The USB shell commands detailed in section 6 are available.

15.3.12 **TLSIM_LEM_Client_PKI_AESGCM_echo.bat**

Connect to LeMonolith in Wi-Fi TLS-IM mode using the AESGCM cipher suite and server authentication based on an X509 certificate. Only an echo operation is supported.

15.3.13 **TLSIM_LEM_Client_PSK_AESCCM_tlsse.bat**

Connect to LeMonolith in Wi-Fi TLS-IM mode using the AESGCM cipher suite and a pre-shared key (PSK). The TLS-SE App commands described in section 8.2 are available.

15.3.14 **TLSIM_SERVER_LOCAL_PSK_AESCCM.bat**

Start a TLS 1.3 server using the AESCCM cipher suite and a pre-shared key (PSK) on localhost (127.0.0.1), TCP port 444.

15.3.15 **TLSIM_SERVER_LEM_CONNECT_REMOTE**

Connect to the local TLS 1.3 PSK server (127.0.0.1:444) using the remote smartcard reader and the TLS-IM0 application hosted in LeMonolith.

15.3.16 **TLSIM_SERVER_LEM_CONNECT_PCSC.bat**

In **USB mode**, connect to the local TLS 1.3 PSK server (127.0.0.1:444) using the TLS-IM module and PC/SC emulation provided by the winscard.dll SHIM.

15.3.17 **TLSIM_SERVER_LEM_CONNECT_SERIAL.bat**

In **USB mode**, **connect** to the local TLS 1.3 PSK server (127.0.0.1:444) using the TLS-IM module through the serial interface.

15.4 **USB BLUETOOTH Tests**

Select the USB_BLUETOOTH mode

Go to the /config directory.

Start USB_TRANS.bat, which is an example of SEPOLIA (Ethereum) transaction generation.

15.5 **TLS_SE_305 Tests**

These tests run in **Wi-Fi mode**; APDU operations require Wi-Fi TLS-IM mode.

Scripts are located in the /TestTLSSE directory

Documentation is located in the /doc directory.

16 Secure Element Certification Procedure over Wi-Fi

Select the Wi-Fi mode.

16.1 Loading the certification authority key

16.1.1 TLS-IM Smartcard

Go to the CertPCSC directory and run *init_ca_key_3.bat* to load the CA public/private keys into the TLS-IM smartcard.

16.1.2 TLS-IM MFA Token

Go to the /CertSerial directory and run *init_ca_key_3.bat* to load the CA public/private keys into the TLS-IM MFA token.

16.2 SE_NET_Cert_SOFT.bat

This script generates a certificate for LeMonolith, with software credentials.

16.3 SE_NET_Cert_SC.bat

This script generates a certificate for LeMonolith, with the TLS-IM smartcard.

16.4 SE_NET_Cert_MFA.bat

This script generates a certificate for LeMonolith, with TLS-IM MFA token

17 SE Authentication Session Procedure (ASP) over Wi-Fi

Select the Wi-Fi mode.

17.1 SE_NET_auth_SOFT.bat

This script opens an authenticated session with LeMonolith.

17.2 SE_NET_auth_SC.bat

This script opens an authenticated session with LeMonolith and requires a TLS-TM smartcard.

17.3 SE_NET_auth_MFA.bat

This script opens an authenticated session with LeMonolith and requires a TLS-TM MFA token.

18 IoSE Tests

Select the USB mode.

The Internet of Secure Elements (IoSE) server starts two TCP daemons, RACS on port 7777 and TLS on port 8888. It uses LeMonolith as TLS-SE TLS1.3 PSK server, identified by the server name COMX001.

APP	APP	APP	APP	GPShell	Terminal		LeMonolith
TLS PSK	APDU	CMD SHELL	TLS Client	APDU	CMD SHELL	USB	
				winscard.dll		winscard.dll	
RFCOMM			TCP/IP	SERIAL		RACS	TLS
Bluetooth			Wi-Fi	USB		TCP/IP (IoSE)	
LeMonolith							

18.1 IOSE_Server_WIN32.bat

This script starts the IoSE server for Windows.

18.2 IOSE_Server_Console.bat

This script starts the IoSE server in console mode.

18.3 IOSE_RACS_List.bat

This script lists the secure elements connected to the IoSE server. LeMonolith is identified by two SEIDs: 0 and 999 (default).

18.4 IOSE_RACS_Console

This script starts a RACS console.

18.5 IOSE_GP_list.bat

This script lists applications stored in the Java Card.

18.6 IOSE_GP_delete

This script deletes the TLS-SE application stored in the Java Card.

18.7 IOSE_GP_install

This script installs the TLS-SE application in the Java Card.

18.8 IOSE_Openssl.bat

This script opens a TLS-PSK session with OPENSSL (127.0.0.1:8888).

18.9 IOSE_KEYSTORE_test_sign.bat

This script runs an ECDSA signature test over TLS using the key at index 0.

18.10 IOSE_Cert_SOFT.bat

This script generates a certificate for SE with software credentials.

18.11 IOSE_Cert_SC.bat

This script generates a certificate for SE with a TLS-IM smartcard.

18.12 IOSE_Cert_MFA.bat

This script generates a certificate for SE with a TLS-IM MFA token.

18.13 IOSE_auth_SOFT.bat

This script opens an authenticated session with LeMonolith.

18.14 IOSE_auth_SC.bat

This script opens an authenticated session with LeMonolith and requires a TLS-IM smartcard.

18.15 IOSE_auth_MFA.bat

This script opens an authenticated session with LeMonolith and requires a TLS-IM MFA token.

19 Ethereum Transactions over Wi-Fi

Select the Wi-Fi mode.

To understand the *Ethereum API* and get a free token, visit: <https://etherscan.io/apis>.

19.1 Ethereum transaction parameters

In file ./MAKE.bat

```
REM ETHEREUM TRANSACTION MAIN PARAMETERS
```

```
set GASPRICE=10
```

```
set APISERVER=api-sepolia.etherscan.io
```

```
set ETHSERVER=sepolia.etherscan.io
```

```
set TOKEN=0
```

```
set NETID=11155111
```

```
set ETHADR=62A52AC04BFB83723FF11295763E93B89D5DCB74
```

```
set ETHKEY=924121A5AAC0FAB04215B4A964D24681ACEC5D66ED61CD34F7770DAA37633F35
```

```
set ETHDATA="hello world"
```

19.2 ETH_gasview.bat

This script opens <https://owlracle.info/sepolia> to display the SEPOLIA gas price.

19.3 ETH_NET_Make_Transaction.bat

This script makes a SEPOLIA transaction.

19.4 ETH_Transaction_Send.bat

This script sends a SEPOLIA transaction.

19.5 ETH_Transaction_View.bat

This script displays the latest SEPOLIA transaction.

20 Certificate Generation

20.1 The fake private key technique

Small and medium enterprises (SMEs) often face a difficult cyber-security reality: limited budgets, legacy systems, and PKI tools that cannot easily be modified.

One practical way to strengthen certificate authority security at low cost is to separate certificate construction from the use of the real CA private key.

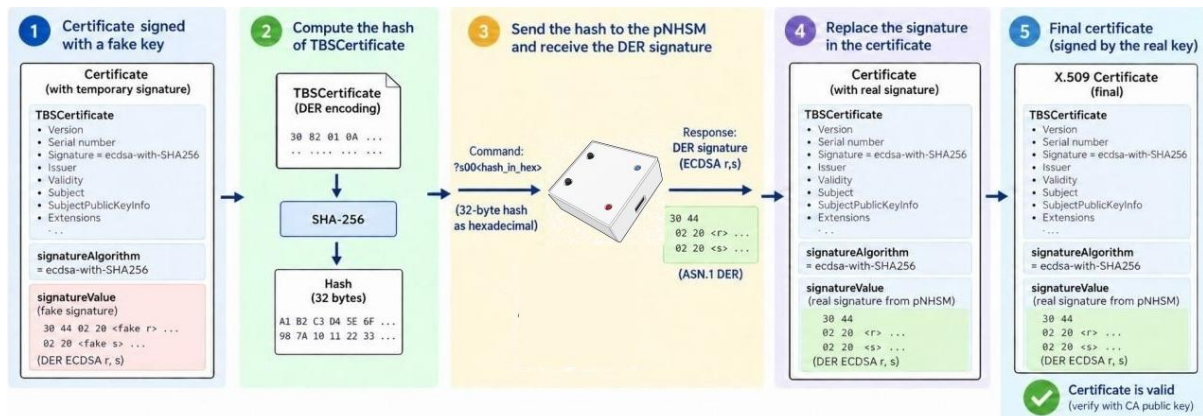
The PKI software first builds the X.509 certificate and signs it with a temporary or "fake" EC private key.

Fake private keys are used in data centers to support HSMs (see, for example, docs.aws.amazon.com/cloudhsm/latest/userguide/key_mgmt_util-getCaviumPrivKey.html).

Once the *TBSCertificate* or *Certificate Signing Request* (CSR) structure is finalized, its DER encoding is hashed with SHA-256. Only this 32-byte hash is sent to a pNHSM, where the real CA private key remains protected.

LeMonolith generates the ECDSA signature and returns the DER-encoded (r,s) signature. The temporary signature in the certificate or CSR is then replaced by the signature returned by LeMonolith.

This approach has several advantages: it can be integrated around existing PKI software without deeply modifying it, it keeps the real CA private key outside the host system, and it significantly reduces the risk of private-key theft or exposure.



Three procedures are used:

- *makek* (*makek.c*): builds a PEM file containing a public key and a fake private key for signing an X509 certificate or *Certificate Signing Request* (CSR) with OpenSSL.
- *extract* (*extract.c*): reads the SHA256 hash from an X509 certificate or CSR.
- *sig* (*sig.c*): replaces the SHA256 hash in an X509 certificate or CSR.

LeMonolith signs the hash of the X509 certificate or CSR.

20.2 CERT_NET_MAKECA.bat

This script generates a public/private key pair on the SECP256r1 prime curve in LeMonolith. It builds a PEM key file containing the correct private key and a fake private key, then generates a self-signed CA certificate using LeMonolith.

20.3 CERT_NET_MAKECSR.bat

This script generates a public/private key pair, then a *Certificate Signing Request* (CSR), using OpenSSL.

20.4 CERT_NET_MAKECSR_TLSSE

This script generates a public/private key pair within LeMonolith, then a *Certificate Signing Request* (CSR) signed by the private key stored in LeMonolith.

20.5 CERT_NET_MAKECERT.bat

This script generates a CA-signed certificate from a CSR using a fake private key. It then extracts the hash, sends it to LeMonolith for signing, and replaces the signature in the certificate. It uses OpenSSL to communicate with LeMonolith.

20.6 CERT_NET_MAKECERT2.bat

This script generates a CA-signed certificate from a CSR using a fake private key. It then extracts the hash, sends it to LeMonolith for signing, and replaces the signature in the certificate. It uses TLSSE to communicate with LeMonolith.

21 Software

21.1 Software components

Software components are located in the `./ESP32Loader/monolith` directory.

- **Arduino IDE 1.8.9**
- **Select the board:** WEMOS D1 MINI ESP32
- **Sketch:** monolith2.ino
- **Dedicated Libraries:** ScLib5c, Cryptoecc, ripemd160, sha256, btools
- **Imported Libraries:** BigNumber
- **Arduino Standard Libraries:** WiFi, EEPROM Crypto

21.2 How to build LeMonolith

- Copy ScLib5c, Cryptoecc, ripemd160, sha256, btools, and BigNumber into the Arduino library directory.
- Compile monolith2.ino. A library-not-found error for ScLib5c.a is expected.
- Copy the ScLib5c file located in /ScLib5c/src/esp32 into the Arduino build directory (specified by build.path= in the Arduino preferences.txt file).
- Compile monolith2.ino again. No error should be reported.

22 Online technical resources

22.1 TLS for Secure Element, TLS-SE

IETF draft TLS For Secure Element, <https://datatracker.ietf.org/doc/html/draft-urien-tls-se-08>

22.2 TLS for secure element input output TLS-SE-IO

IETF draft TLS for Secure Element Input Output, <https://datatracker.ietf.org/doc/html/draft-urien-core-tls-se-io-02>

22.3 TLS identity module, TLS-IM

IETF draft TLS Identity Module, <https://datatracker.ietf.org/doc/html/draft-urien-tls-im-10>

22.4 Remote APDU Server (RACS)

IETF Draft, Remote APDU Call Secure, <https://datatracker.ietf.org/doc/html/draft-urien-core-racs-19>

22.5 TLS for secure element Rendez-Vous TLS-SE-RDV

IETF draft, TLS For Secure Element Rendez Vous, <https://datatracker.ietf.org/doc/draft-urien-tls-se-rdv/>

22.6 Remote APDU Call Secure Lite(RACSL)

Remote APDU Call Secure Lite(RACSL), <https://datatracker.ietf.org/doc/draft-urien-core-racsl/>

23 ANNEXE

23.1 The Crypto Currency (CC) Application

The Crypto Currency application (CCSC), with AID 010203040601, has three PINs: administrator, user, and user2. The defaults are eight zeros (3030303030303030) for the administrator and four zeros (30303030) for user and user2.

The application can generate, derive according to BIP32, or import up to 16 elliptic curve keys. These keys generate ECDSA signatures used by the Bitcoin and Ethereum crypto currencies.

A 16 KB area of read/write nonvolatile memory, protected by a dedicated PIN (User2), is available for storing sensitive information.

23.1.1 The Select Command

This command starts the Crypto Currency smartcard application.
On success, it returns the status word SW1 SW2 = 9000.

Command

CLA	INS	P1	P2	P3	AID
00	A4	04	00	06	010203040601

Response

SW1	SW2
90	00

23.1.2 The Verify UserPin Command

This command verifies the user PIN. UserPin is required for signature operations.

On success, it returns the status word SW1 SW2 = 9000.
Otherwise, it returns SW1=63, SW2=number of remaining tries (at most 3).

Command

CLA	INS	P1	P2	P3	UserPin
00	20	00	00	04	3030303030

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1P2

23.1.3 The Verify UserPin2 Command

This command verifies the second user PIN. UserPin2 is required for memory read and write operations.

On success, it returns the status word SW1 SW2 = 9000.

Otherwise, it returns SW1=63, SW2=number of remaining tries (at most 3).

Command

CLA	INS	P1	P2	P3	UserPin2
00	20	00	02	04	3030303030

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1P2

23.1.4 The Verify AdminPin command

This command verifies the administrator PIN and provides access to all features of the crypto currency application. If P2 is set to FF, UserPin and UserPin2 are reset to their default value (four zeros).

On success, it returns the status word SW1 SW2 = 9000.

Otherwise, it returns SW1=63, SW2=number of remaining tries (at most ten).

Command

CLA	INS	P1	P2	P3	AdminPin
00	20	00	01	08	303030303030303030
			FF Reset to default		

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1P2

23.1.5 The ChangePin command

This command sets a PIN (UserPin, UserPin2, AdminPin) to a new value. P2 is set to 00, 02, or 01 for UserPin, UserPin2, or AdminPin, respectively. On success, it returns the status word SW1, SW2 = 9000. Otherwise, it returns SW1=63, SW2=number of remaining tries.

Command

CLA	INS	P1	P2	P3	OldPin	NewPIN
00	24	00	00	10	30303030FFFFFFFF	31313131FFFFFFFF
00	24	00	02	10	30303030FFFFFFFF	30303030FFFFFFFF
00	24	00	01	10	3030303030303030	3131313131313131

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1P2

23.1.6 The GetStatus command

This command returns the current state of the crypto currency application. At least one PIN (UserPin, UserPin2, or AdminPin) must have been verified beforehand.

Command

CLA	INS	P1	P2	P3
00	87	00	00	0A

Response

SW1	SW2	Comment
90	00	Success
63	80	PIN required

This command returns 10 bytes.

byte0: b0= ECDSA, b1=DH, b3=SHA256, b4=HMAC-512, b5= BigInteger (OK= 0x07)
byte1: The maximum number of keys supported by the crypto currency application (16).
byte2, byte3: The CCSC application version (0x0007 = v0.7).
byte4, byte5: The size of the user memory (for example 4000 for 16 KB)
byte6, byte7: 16 bits (b15...b1b0) indicating which keys are defined (bit bi identifies key i); for example, 0003 indicates key1 (bit1) and key0 (bit0).
byte8, byte9: 16 bits (b15...b1b0) indicating which key trees are defined (bit bi identifies tree i); for example, 0003 indicates tree1 (bit1) and tree0 (bit0).

23.1.7 The Write Command

This command writes data to nonvolatile memory. It can securely store information in the address range [0000, 0C00].

UserPin2 or AdminPin must have been verified beforehand.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	Data
00	D0	AdrMSB	AdrLSB	Data Length	Data to be written

The starting address, ranging from 0000 to 10FF, is encoded in two bytes (P1, P2), where P1 is the *most significant byte* (MSB) and P2 is the *least significant byte* (LSB).

Address Mapping

Start Address	Length	Comment	PIN required
0000	0C00	Data Area	User2 or Admin
0C00	0400	Key Dump Area	Admin
1000	0100	Key Label - 32 bytes/key	Admin

Response

SW1	SW2	comment
90	00	OK
63	80	PIN required
6D	02	Invalid Address

23.1.8 The Read Command

This command reads data from nonvolatile memory.

UserPin2 or AdminPin must have been verified beforehand.

On success, it returns the status word SW1,SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3
00	B0	AdrMSB	AdrLSB	Length to be read

The starting address, ranging from 0000 to 10FF, is encoded in two bytes (P1, P2), where P1 is the most significant byte (MSB) and P2 is the least significant byte (LSB).

Address Mapping

Start Address	Length	Comment	PIN required
0000	0C00	Data Area	User2 or Admin
0C00	0400	Key Dump Area	Admin
1000	0100	Key Label - 32 bytes/key	Admin

Response

Body	SW1	SW2	comment
Data	90	00	OK
Empty	63	80	PIN required
Empty	6D	01	Invalid Address

23.1.9 The Clear KeyPair & InitCurve Command

This command **MUST** be used before any key setting or key generation operation.

This command clears the curve parameters.

The InitCurve command is required to configure the elliptic curve, unless the InitCurve option is used.

The Admin PIN is required, except for P1=10 (Reset Key Tree), which accepts either the User or Admin PIN.

The P1=80 option clears either the public or private key and initializes the associated curve. Examples:

- P1=C0, clear public key and initialize curve SECP256k1.
- P1=A0, clear private key and initialize curve SECP256k1.

The P1=10 option resets and initializes a KeyTree.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	81	00 – Clear Keys 10 – Clear Keys & KeyTree	Key Index [0,15]	00	Admin
00	81	SECP256k1 80 – Clear Keys & InitCurve 40 - Clear Public Key Only 20 - Clear Private Key Only 10 - Reset KeyTree	Key Index [0,15]	00	Admin

Response

SW1	SW2	Comment
90	00	Key Reset Done
63	80	PIN required
69	85	Bad index

23.1.10 The InitCurve & InitTree Command

This command initializes the elliptic curve parameters and optionally a KeyTree.

The KeyTree is initialized from a seed according to the BIP32 specification. Two seed generation modes are available:

- The seed value is imported;
- The seed value is randomly generated.

The keys **MUST** be cleared before this operation.

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	89	00 - SECP256k1	Key Index [0,15]	00	Admin
00	89	00 - SECP256k1	Tree Index [0,15]	Seed Length If P3=1, the payload is the size of the seed to be randomly generated	Admin

Response

Data	SW1	SW2	Comment
	90	00	Key Reset Done
TreeStatus 2 bytes b _i = TreeIndex	90	00	KeyTree initialized
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is defined
	64	02	Private Key is defined
	6A	86	Incorrect P1P2
	69	85	KeyTree initialization error

23.1.11 The Generate KeyPair Command

This command generates an elliptic curve public/private key pair or derives a key according to the BIP32 specification.

The keys **MUST** be cleared before this operation.

The Admin PIN is required. For a KeyTree operation (P2 is a KeyTree index and P3#0), either the User PIN or Admin PIN is required.

If P3 is set to zero, a public/private key pair is randomly generated.

If P3 is a multiple of 4, a hardened private key is generated using a derivation path consisting of n 32-bit words, $IH_1/IH_2/\dots/IH_n$. The public key is not computed. The MSB of each 32-bit word IH_i MUST be set.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	82	00	Key Index [0,15]	00	Admin
00	82	00	Tree Index [0,15]	4 n	Admin

Response

SW1	SW2	Comment
90	00	OK
63	80	PIN required
69	85	Bad index
64	01	Public Key is defined
64	02	Private Key is defined
6D	10	Key Generation Error

23.1.12 The Dump KeyPair Command

This command dumps the elliptic curve public and private keys.

It returns the size of the data written in the nonvolatile memory, in the *Key Dump* area whose address starts at 0C00

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	83	00	Key Index [0,15]	02	Admin
00	83	FF Reset DUMP Area	Not Used	00	Admin

Response

Body	SW1	SW2	Comment
Total Length 2 bytes SECP256k1 0185 SECP256v1 0201	90	00	Key Reset Done
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is not defined
	64	02	Private Key is not defined
	6A	86	Incorrect P1 P2

Dump KeyPair: Memory Mapping

Address = 0C00	Comment
Total Length, 2 bytes	
PubKey A Length, 2 bytes	The length of the A parameter
PubKey A parameter value	The value of the A parameter
PubKey B Length, 2 bytes	The length of the B parameter
PubKey B parameter value	The value of the B parameter
PubKey G Length, 2 bytes	The length of the Generator
PubKey G value	The value of the Generator
PubKey R Length, 2 bytes	The length of the R parameter
PubKey R value	The value of the R parameter (Order of the Generator)
PubKey W Length, 2 bytes	The length of the W parameter
PubKey W value	The value of the W Public Key (EC Point)
PubKey Field Length, 2 bytes	The length of the Field parameter
PubKey Field Value	The value of the prime p of the field Z/pZ
PubKey Size, 2 bytes	The size of the Public Key object
PrivKey A Length, 2 bytes	The length of the A parameter
PrivKey A parameter value	The value of the A parameter
PrivKey B Length, 2 bytes	The length of the B parameter
PrivKey B parameter value	The value of the B parameter
PrivKey G Length, 2 bytes	The length of the Generator
PrivKey G value	The value of the Generator
PrivKey R Length, 2 bytes	The length of the R parameter
PrivKey R value	The value of the R parameter (Order of the Generator)
PrivKey S Length, 2 bytes	The length of the S parameter
PrivKey S value	The value of the S, Private Key (32 bytes)
PrivKey Field Length, 2 bytes	The length of the Field parameter
PrivKey Field Value	The value of the prime p of the field (Z/pZ)
PrivKey Size, 2 bytes	The size of the Private Key object

23.1.13 The GetInfo command

This command retrieves information for a given key index, including the key label, tree seed, and private key.

It requires the Admin PIN, or the user PIN for the *CardContent* option.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	86	00	Key Index [0,15]	00	Admin

CLA	INS	P1	P2	P3	Payload	PIN required
00	86	FF CardContent	00	0 to 32	xx random bytes (r)	User or Admin

Response

Body	SW1	SW2	Comment
If P1=0 Status 2 bytes 0x0000: no tree or key 0x0001: Tree 0x0002: Key 0x0003: Tree & Key Label Length (2 bytes) Label Value TreeSeed Length (2 bytes) TreeSeed value Private Key Length (2 bytes) Private key value	90	00	If status # 0 (tree or key) If tree If key
If P1=FF hash length 2 bytes hash value signature length 2 bytes signature value (ASN.1 encoded)	6C 90	xx 00	Read the signed card content sign(sha256(hash-value r))
	63	80	PIN required
	69	85	Bad index

23.1.14 The Get KeyParameter Command

This command retrieves the elliptic curve public and private key parameters.

It requires the User or Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	84	00 Parameter A	Key Index [0,15]	00	Admin or User
		01 Parameter B		00	Admin or User
		02 Parameter Field (Z/pZ)		00	Admin or User
		03 Parameter G (generator)		00	Admin or User
		04 Parameter K (cofactor)		00	Admin or User
		05 Parameter R (order of G)		00	Admin or User
		06 Parameter W (Public Key)		43	Admin or User
		07 Parameter S (Private Key)		22	Admin
		08 Key Label		20	Admin or User
		09 x value of the public key		00	Admin or User
		0A TreeSeed		00	Admin

Response

Body	SW1	SW2	Comment
Param0 Length – Param0 value Param1 Length – Param1 value Param2 Length – Param2 value Param3 Length – Param3 value Param4 Length – Param4 value Param5 Length – Param5 value Param6 Length – Param6 value Param7 Length – Param7 value	90	00	The response includes a 2-byte length field for parameters 0 to 7.
Param8 (Key Label) Value	90	00	OK
Param9 Length – Param9 value	90	00	The x value of the public key
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is not defined
	64	02	Private Key is not defined
	6A	86	Incorrect P1P2
	6D	30	Javacard Exception
ParamA Length – ParamA value	90	00	OK

23.1.15 The Set KeyParameter Command

This command sets elliptic curve parameters, including public and private key parameters. The Admin PIN is required, except for Parameter 6 (Public Key), which accepts either the User or Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	88	00 Parameter A	Key Index [0,15]	Length	Admin
		01 Parameter B		Length	Admin
		02 Parameter Field (Z/pZ)		Length	Admin
		03 Parameter G (generator)		Length	Admin
		04 Parameter K (cofactor)		Length	Admin
		05 Parameter R (order of G)		Length	Admin
		06 Parameter W (Public Key) The public key is checked against the private key and reset if an error occurs.		41	Admin
		07 Parameter S (Private Key) The private key MUST be initialized before setting the public key		20	Admin
		08 Key Label		20	Admin

Response

SW1	SW2	Comment
90	00	OK
63	80	PIN required
69	85	Bad index
64	01	Public Key is defined
64	02	Private Key is defined
6A	86	Incorrect P1P2
6D	40	Javacard Exception

23.1.16 The SignECDSA command

This command generates an ECDSA signature.

It requires the AdminPin or UserPin.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	Data	PIN required
00	80	00 Signature without digest	Key Index [0,15]	Length 20	Data to be signed	Admin or User
00	80	21 Signature with SHA256	Key Index [0,15]	Length	Data to be hashed and signed	Admin or User

Response

Body	SW1	SW2	Comment
Length (2 bytes) ASN.1 ECDSA Signature Encoding	90	00	OK
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is not defined
	64	02	Private Key is not defined
	6A	86	Incorrect P1P2
	6D	20	Signature Error

23.1.17 The GetCertificate command

This command reads the card certificate.

It requires the Admin or User PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

The card certificate is an ECDSA SHA256 signature, over the secp56k1 curve, of the public key at index 0 (in full representation, 65 bytes). It consists of two concatenated 32-byte integers, r and s.

The experimental Ethertrust public key, on the curve secp256k1, is:

04

6099836D971593AAA2C1C32B6DB9EF9521041795E21CF1E7511DF3BD358F97DF
358B33A875E359CBE236163D6DBAEDFEC6C9393522C7EBC25A7CC85E1F0A7D67

Command

CLA	INS	P1	P2	P3	PIN required
00	8E	00	00	00	Admin or User

Response

Body	SW1	SW2	Comment
64-byte certificate	90	00	Two 32-byte integers, r and s.
Empty	63	80	PIN required

23.1.18 The SetCertificate command

This command sets the card certificate.

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Two operating modes are supported: importing a 64-byte certificate or computing the certificate internally using an embedded key (self-signing).

Command

CLA	INS	P1	P2	P3	Payload	PIN required
00	8F	00	00	40	64-byte certificate	Admin
00	8F	FF	KeyIndex	00	Empty	Admin

Response

SW1	SW2	Comment
90	00	Certificate loaded or generated
63	80	PIN required
69	84	Certificate Already Set
69	85	Wrong Certificate Length
64	03	Private Key is not set

23.2 The TLS-IM Application

The extended TLS Identity Module application, TLS-IM version 11c, with AID 010203040700, has two PINs: administrator and user. The defaults are eight zeros (3030303030303030) for the administrator and four zeros (30303030) for the user.

Four P256 elliptic curve key pairs are available for ECDSA signatures.

The application securely performs the procedures used for TLS 1.3 PSK authentication.

A 1 KB area of read/write nonvolatile memory, protected by a dedicated PIN (User), is available for storing X509 certificates or other data.

23.2.1 The Select Command

This command starts the Crypto Currency smartcard application. On success, it returns the status word SW1 SW2 = 9000.

Command

CLA	INS	P1	P2	P3	AID
00	A4	04	00	06	010203040700

Response

SW1	SW2
90	00

23.2.2 The Verify UserPin Command

This command verifies the user PIN. UserPin is required for signature operations.

On success, it returns the status word SW1 SW2 = 9000. Otherwise, it returns SW1=63, SW2=number of remaining tries (at most 3).

Command

CLA	INS	P1	P2	P3	UserPin
00	20	00	00	04	3030303030

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1 P2

23.2.3 The Verify AdminPin command

This command verifies the administrator PIN and provides access to all features of the crypto currency application. If P2 is set to FF, UserPin is reset to its default value (four zeros).

On success, it returns the status word SW1 SW2 = 9000.

Otherwise, it returns SW1=63, SW2=number of remaining tries (at most ten).

Command

CLA	INS	P1	P2	P3	AdminPin
00	20	00	01	08	303030303030303030
			FF Reset to default		

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1 P2

23.2.4 The ChangePin command

This command changes a PIN (UserPin or AdminPin) to a new value.

P2 is set to 00 or 01 for UserPin or AdminPin, respectively.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=number of remaining tries.

Command

CLA	INS	P1	P2	P3	OldPin	NewPIN
00	24	00	00	10	30303030FFFFFFF	31313131FFFFFFF
00	24	00	01	10	3030303030303030	3131313131313131

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1P2

23.2.5 The GetStatus command

This command returns the current state of the crypto currency application. At least one PIN (UserPin or AdminPin) must have been verified beforehand.

Command

CLA	INS	P1	P2	P3
00	87	00	00	0A

Response

SW1	SW2	Comment
90	00	Success
63	80	PIN required

This command returns 4 bytes.

b0=0,b1=2, version=b2.b3 (1.1)

23.2.6 The Write Command

This command writes data to nonvolatile memory. It is used to store certificates in the address range [0000, 03FF].

UserPin or AdminPin must have been verified beforehand.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	Data
00	D0	AdrMSB	AdrLSB	Data Length	Data to be written

The starting address, ranging from 0000 to 03FF, is encoded in two bytes (P1, P2), where P1 is the *most significant byte* (MSB) and P2 is the *least significant byte* (LSB).

Address Mapping

Start Address	Length	Comment	PIN required
0000	0400	Data Area	User or Admin

Response

SW1	SW2	comment
90	00	OK
63	80	PIN required
6D	02	Invalid Address

23.2.7 The Read Command

This command reads data from nonvolatile memory.

UserPin or AdminPin must have been verified beforehand.

On success, it returns the status word SW1,SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3
00	B0	AdrMSB	AdrLSB	Length to be read

The starting address, ranging from 0000 to 03FF, is encoded in two bytes (P1, P2), where P1 is the most significant byte (MSB) and P2 is the least significant byte (LSB).

Address Mapping

Start Address	Length	Comment	PIN required
0000	0400	Data Area	User or Admin

Response

Body	SW1	SW2	comment
Data	90	00	OK
Empty	63	80	PIN required
Empty	6D	01	Invalid Address

23.2.8 The Clear KeyPair command

This command MUST be used before any key setting or key generation operation.

This command clears the curve parameters.

The InitCurve command is needed to configure the Elliptic Curve

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	81	00	Key Index [0,3]	00	Admin

Response

SW1	SW2	Comment
90	00	Key Reset Done
63	80	PIN required
69	85	Bad index

23.2.9 The InitCurve command

This command initializes the SECP256r1 (P256) elliptic curve parameters.

The keys MUST be cleared before this operation.

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	89	01 P256	Key Index [0,3]	00	Admin

Response

Data	SW1	SW2	Comment
	90	00	Key Reset Done
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is defined
	64	02	Private Key is defined
	6A	86	Incorrect P1P2

23.2.10 The Generate KeyPair Command

The keys MUST be cleared before this operation.

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	82	00	Key Index [0,3]	00	Admin

Response

SW1	SW2	Comment
90	00	OK
63	80	PIN required
69	85	Bad index
64	01	Public Key is defined
64	02	Private Key is defined
6D	10	Key Generation Error

23.2.11 The Get KeyParameter Command

This command retrieves the elliptic curve public and private key parameters.

It requires the User or Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	84	00 Parameter A	Key Index [0,3] FE for Cert Key FF for Ephemeral Key	00	Admin or User
		01 Parameter B		00	Admin or User
		02 Parameter Field (Z/pZ)		00	Admin or User
		03 Parameter G (generator)		00	Admin or User
		04 Parameter K (cofactor)		00	Admin or User
		05 Parameter R (order of G)		00	Admin or User
		06 Parameter W (Public Key)		43	Admin or User
		07 Parameter S (Private Key)		22	Admin

Response

Body	SW1	SW2	Comment
Param0 Length – Param0 value Param1 Length – Param1 value Param2 Length – Param2 value Param3 Length – Param3 value Param4 Length – Param4 value Param5 Length – Param5 value Param6 Length – Param6 value Param7 Length – Param7 value	90	00	The response includes a 2-byte length field.
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is not defined
	64	02	Private Key is not defined
	6A	86	Incorrect P1P2
	6D	30	Javacard Exception
ParamA Length – ParamA value	90	00	OK

23.2.12 The Set KeyParameter Command

This command sets elliptic curve parameters, including public and private key parameters. The Admin PIN is required, except for Parameter 6 (Public Key), which accepts either the User or Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	88	00 Parameter A	Key Index [0,3]	Length	Admin
		01 Parameter B		Length	Admin
		02 Parameter Field (Z/pZ)		Length	Admin
		03 Parameter G (generator)		Length	Admin
		04 Parameter K (cofactor)		Length	Admin
		05 Parameter R (order of G)		Length	Admin
		06 Parameter W (Public Key) The public key is checked against the private key and reset if an error occurs.		41	Admin
		07 Parameter S (Private Key) The private key MUST be initialized before setting the public key		20	Admin

Response

SW1	SW2	Comment
90	00	OK
63	80	PIN required
69	85	Bad index
64	01	Public Key is defined
64	02	Private Key is defined
6A	86	Incorrect P1P2
6D	40	Javacard Exception

23.2.13 The SignECDSA command

This command generates an ECDSA signature.

It requires the AdminPin or UserPin.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	Data	PIN required
00	80	00 Raw Signature	Key Index [0,3] FE= Cert Key	Length 20	Data to be signed	Admin or User
00	80	21 Signature with SHA256	Key Index [0,3] FE= Cert Key	Length	Data to be hashed and signed	Admin or User

Response

Body	SW1	SW2	Comment
Length (2 bytes) ASN.1 ECDSA Signature Encoding	90	00	OK
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is not defined
	64	02	Private Key is not defined
	6A	86	Incorrect P1P2
	6D	20	Signature Error

23.2.14 The Diffie Hellman Command

This command computes a Diffie-Hellman (DH) secret and optionally generates an ephemeral public/private key pair.

The Admin or User PIN is required.

Command

CLA	INS	P1	P2	P3	Data	PIN required
00	8A	01 P256 Curve	Key Index [0,3] FF for internal key generation 8x for key generation for index x	Length 41	Public Key Uncompressed format (04)	Yes Admin User

Response

Body= Empty or 32 bytes DH	SW1	SW2	Comment
	90	00	No Error
	6D	50	DH Error

23.2.15 The GenerateRandom command

This command generates random bytes.

It requires no PIN.

Command

CLA	INS	P1	P2	P3	PIN required
00	8B	00	00	Length	NO

Response

Body= P3 bytes or empty	SW1	SW2	Comment
	90	00	No Error
	69	85	Error

23.2.16 The HMAC Command

This command computes keys used by TLS 1.3. It requires the Admin or User PIN.

Command

CLA	INS	P1	P2	P3	Data	PIN
00	85	00	02= Compute HMAC LengthKey [Key] LengthData [Data]	2+ LengthKey+ LengthData	32 bytes	Admin or User
		00 FF	0A= EXTRACT_EARLY P1=00 return ESK P1=FF return ESK, HSK, eBSK, rBSK, feBSK, frBSK	23	01 00 20 PSK (32 bytes)	Admin or User
		00 01	0B EXPAND_EARLY P1=0 "c e traffic" P1=1 "e exp master"	3+ DataLength	00 20 DataLength Data	Admin or User
		00	0C= HMAC_EBSK	Data Length	Data	Admin or User
		00	0D=HMAC_RBSK	Data Length	Data	Admin or User
		00	0E=EXTRACT_HANDSHAKE (HMAC_HSK)	Data Length	Data (PubKey)	Admin or User

Response

Body 32 bytes 6x32 for EXTRACT_EARLY with P1=FF	SW1	SW2	Comment
	90	00	No Error
	6A	86	Wrong P1P2

23.2.17 The GetCertificate command

This command reads the card certificate.

It requires the Admin or User PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

The card certificate is an ECDSA SHA256 signature, over the secp56k1 curve, of the public key at index 0 (in full representation, 65 bytes). It consists of two concatenated 32-byte integers, r and s.

Command

CLA	INS	P1	P2	P3	PIN required
00	8E	00	00	00	Admin or User

Response

Body	SW1	SW2	Comment
64-byte certificate	90	00	Two 32-byte integers, r and s.
Empty	63	80	PIN required

23.2.18 The SetCertificate command

This command sets the card certificate.

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Two operating modes are supported: importing a 64-byte certificate or computing the certificate internally using an embedded key (self-signing).

Command

CLA	INS	P1	P2	P3	Payload	PIN required
00	8F	00	00	40	64-byte certificate	Admin
00	8F	FF	KeyIndex	00	Empty	Admin

Response

SW1	SW2	Comment
90	00	Certificate loaded or generated
63	80	PIN required
69	84	Certificate Already Set
69	85	Wrong Certificate Length
64	03	Private Key is not set

23.3 The TLS-IM0 Application

The minimal TLS Identity Module application, TLS-IM0, with AID 010203040800, has two PINs: administrator and user. The defaults are eight zeros (3030303030303030) for the administrator and four zeros (30303030) for the user.

The application securely performs the procedures used for TLS 1.3 PSK authentication.

23.3.1 The Select Command

This command starts the Crypto Currency smartcard application.
On success, it returns the status word SW1 SW2 = 9000.

Command

CLA	INS	P1	P2	P3	AID
00	A4	04	00	06	010203040800

Response

SW1	SW2
90	00

23.3.2 The Verify UserPin Command

This command verifies the user PIN. UserPin is required for signature operations.

On success, it returns the status word SW1 SW2 = 9000.
Otherwise, it returns SW1=63, SW2=number of remaining tries (at most 3).

Command

CLA	INS	P1	P2	P3	UserPin
00	20	00	00	04	3030303030

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1 P2

23.3.3 The Verify AdminPin command

This command verifies the administrator PIN and provides access to all features of the crypto currency application. If P2 is set to FF, UserPin is reset to its default value (four zeros).

On success, it returns the status word SW1 SW2 = 9000.
Otherwise, it returns SW1=63, SW2=number of remaining tries (at most ten).

Command

CLA	INS	P1	P2	P3	AdminPin
00	20	00	01 FF Reset to default	08	30303030303030303030

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1 P2

23.3.4 The ChangePin command

This command changes a PIN (UserPin or AdminPin) to a new value.

P2 is set to 00 or 01 for UserPin or AdminPin, respectively.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=number of remaining tries.

Command

CLA	INS	P1	P2	P3	OldPin	NewPIN
00	24	00	00	10	30303030FFFFFFFF	31313131FFFFFFFF
00	24	00	01	10	3030303030303030	3131313131313131

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1P2

23.3.5 The HMAC Command

This command computes keys used by TLS 1.3.

It requires the Admin or User PIN.

Command

CLA	INS	P1	P2	P3	Data	PIN
00	85	00	02= Compute HMAC LengthKey [Key] LengthData [Data]	2+ LengthKey+ LengthData	32 bytes	Admin or User
		00 FF	0A= EXTRACT_EARLY P1=00 return ESK P1=FF return ESK, HSK, eBSK, rBSK, feBSK, frBSK	23	01 00 20 PSK (32 bytes)	Admin or User
		00 01	0B= EXPAND_EARLY P1=0 "c e traffic" P1=1 "e exp master"	3+ DataLength	00 20 DataLength Data	Admin or User
		00	0C= HMAC_EBSK	Data Length	Data	Admin or User
		00	0D=HMAC_RBSK	Data Length	Data	Admin or User
		00	0E=EXTRACT_HANDSHAKE (HMAC_HSK)	Data Length	Data (PubKey)	Admin or User

Response

Body 32 bytes 6x32 for EXTRACT_EARLY with P1=FF	SW1	SW2	Comment
	90	00	No Error
	6A	86	Wrong P1P2

23.4 The TLS-SE Strict Application

This application supports only three APDU commands: RECV, SEND, and SELECT (optional).

23.4.1 The Select Command

This command starts the TLS-SE application.
On success, it returns the status word SW1 SW2 = 9000.

Command

CLA	INS	P1	P2	P3	AID
00	A4	04	00	06	010203040500

Response

SW1	SW2
90	00

23.4.2 The SEND Command

This command writes a TLS packet to the secure element and returns a TLS message.
It requires no PIN.

Command

CLA	INS	P1	P2	P3	Data
00	D8	0 Normal Operation 1 if tls is open: Decrypt 2 if tls is open: Encrypt 3 if tls is open: Echo FF Rx Echo 1 in ClientHello PSK test 81 in ClientHelloPKI test	0:more 1: first 2: last 3: first&last	Data Length 00: Start/Reset TLS session	Data to be written

Response

Body	SW1	SW2	Comment
Optional message	90	00	OK
Optional message	90	01	TLS channel is open
Optional message	90	02	TLS channel is closed
Optional message	61	length	More data (length bytes) to read
Optional message	9F	length	More data (length bytes) to read
Empty	6D	02	Write Error
Empty	6D	01	Read Error
Empty	6D	14	Decryption Error
Empty	6D	15	Encryption Error
Empty	6D	16	TLS Error

23.4.3 The RECV Command

This command reads a TLS packet.
It requires no PIN.

Command

CLA	INS	P1	P2	P3
00	C0	0	0	Data Length

Response

Body	SW1	SW2	Comment
TLS Packet	90	00	No more data to read
TLS Packet	61	Length	More data (length bytes) to read
TLS Packet	9F	Length	More data (length bytes) to read

23.5 The TLS-SE Combi Application

This application is intended for testing. Unlike the strict TLS-SE application, which supports only RECV, SEND, and SELECT (optional), it provides debugging facilities and TLS-IM features.

23.5.1 The Select Command

This command starts the TLS-SE application.
On success, it returns the status word SW1 SW2 = 9000.

Command

CLA	INS	P1	P2	P3	AID
00	A4	04	00	06	010203040500

Response

SW1	SW2
90	00

23.5.2 The Verify UserPin Command

This command verifies the user PIN. UserPin is required for signature operations.

On success, it returns the status word SW1 SW2 = 9000.
Otherwise, it returns SW1=63, SW2=number of remaining tries (at most 3).

Command

CLA	INS	P1	P2	P3	UserPin
00	20	00	00	04	3030303030

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1P2

23.5.3 The Verify AdminPin command

This command verifies the administrator PIN and provides access to all features of the crypto currency application. If P2 is set to FF, UserPin is reset to its default value (four zeros).

On success, it returns the status word SW1 SW2 = 9000.
Otherwise, it returns SW1=63, SW2=number of remaining tries (at most ten).

Command

CLA	INS	P1	P2	P3	AdminPin
00	20	00	01 FF Reset to default	08	30303030303030303030

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1P2

23.5.4 The ChangePin command

This command changes a PIN (UserPin or AdminPin) to a new value.

P2 is set to 00 or 01 for UserPin or AdminPin, respectively.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=number of remaining tries.

Command

CLA	INS	P1	P2	P3	OldPin	NewPIN
00	24	00	00	10	30303030FFFFFFFF	31313131FFFFFFFF
00	24	00	01	10	3030303030303030	3131313131313131

Response

SW1	SW2	Comment
90	00	Success
63	Number of remaining tries	Fail
67	00	Wrong Length
6B	00	Wrong P1P2

23.5.5 The GetStatus command

This command returns the current state of the crypto currency application. At least one PIN (UserPin or AdminPin) must have been verified beforehand.

Command

CLA	INS	P1	P2	P3
00	87	00	00	0A

Response

SW1	SW2	Comment
90	00	Success
63	80	PIN required

This command returns 8 bytes.

B0 B1 B2 B3: version= 3.0.3

B4 B5: RAM_SIZE MSB RAM_SIZE LSB (initial)

B6 B7: RAM_SIZE MSB RAM_SIZE LSB (remaining)

23.5.6 The Write Command

This command writes data to nonvolatile memory in the address range [0000, 13FF] (5120 bytes).

UserPin or AdminPin must have been verified beforehand.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	Data
00	D0	AdrMSB	AdrLSB	Data Length	Data to be written

The starting address, ranging from 0000 to 13FF, is encoded in two bytes (P1, P2), where P1 is the *most significant byte* (MSB) and P2 is the *least significant byte* (LSB).

Address Mapping

Start Address	Length	Comment	PIN required
0000	0800	Data Area	User or Admin

Response

SW1	SW2	comment
90	00	OK
63	80	PIN required
6D	02	Invalid Address

23.5.7 The Read Command

This command reads data from nonvolatile memory.

UserPin or AdminPin must have been verified beforehand.

On success, it returns the status word SW1,SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3
00	B0	AdrMSB	AdrLSB	Length to be read

The starting address, ranging from 0000 to 13FF (5120 bytes), is encoded in two bytes (P1, P2), where P1 is the most significant byte (MSB) and P2 is the least significant byte (LSB).

Address Mapping

Start Address	Length	Comment	PIN required
0000	0800	Data Area	User or Admin

Response

Body	SW1	SW2	comment
Data	90	00	OK
Empty	63	80	PIN required
Empty	6D	01	Invalid Address

23.5.8 The Clear KeyPair command

This command MUST be used before any key setting or key generation operation.

This command clears the curve parameters.

The InitCurve command is required to configure the Elliptic Curve

It clears RSA20248 key or EC256 key.

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	81	00	AlgoId b5 b5=0 EC256, b5=1 RSA2048 Key Index b1b0	00	Admin

Response

SW1	SW2	Comment
90	00	Key Reset Done
63	80	PIN required
69	85	Bad index

23.5.9 The InitCurve command

This command initializes the elliptic curve parameters for SECP256r1 (P256) by default, or for SECP256k1.

The keys MUST be cleared before this operation.

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	89	FF= SECP256k1 other=SECP256 prime	Key Index [0,3]	00	Admin

Response

Data	SW1	SW2	Comment
	90	00	Key Reset Done
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is defined
	64	02	Private Key is defined
	6A	86	Incorrect P1P2

23.5.10 The Generate KeyPair Command

The keys MUST be cleared before this operation.

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	82	00	b5 AlgoId b5=0 EC256, b5=1 RSA2048 b1b0 :Key Index [0,3]	00	Admin

Response

SW1	SW2	Comment
90	00	OK
63	80	PIN required
69	85	Bad index
64	01	Public Key is defined
64	02	Private Key is defined
6D	10	Key Generation Error

23.5.11 The Get KeyParameter Command

This command retrieves the public and private key parameters for EC256 or RSA 2048.

It requires the User or Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command for EC256

CLA	INS	P1	P2	P3	PIN required
00	84	00 Parameter A	b5=0, EC256 b1b0 :Key Index Ephemeral Key FF Cert Key FE	00	Admin or User
		01 Parameter B		00	Admin or User
		02 Parameter Field (Z/pZ)		00	Admin or User
		03 Parameter G (generator)		00	Admin or User
		04 Parameter K (cofactor)		00	Admin or User
		05 Parameter R (order of G)		00	Admin or User
		06 Parameter W (Public Key)		43	Admin or User
		07 Parameter S (Private Key)		22	Admin Forbidden for KeyIndex FE

Response for EC256

Body	SW1	SW2	Comment
Param0 Length – Param0 value Param1 Length – Param1 value Param2 Length – Param2 value Param3 Length – Param3 value Param4 Length – Param4 value Param5 Length – Param5 value Param6 Length – Param6 value Param7 Length – Param7 value	90	00	The response includes a 2-byte length field.
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is not defined
	64	02	Private Key is not defined
	6A	86	Incorrect P1P2
	6D	30	Javacard Exception
ParamA Length – ParamA value	90	00	OK

Command for RSA2048

CLA	INS	P1	P2	P3	PIN required
00	84	00 Modulo	b5=1, RSA2048 b1b0 :Key Index	00	Admin or User
		01 Public Exponent		00	Admin or User
		02 P		00	Admin or User
		03 Q		00	Admin
		04 DP1		00	Admin
		05 DQ1		00	Admin
		06 PQ		00	Admin

Response for EC256

Body	SW1	SW2	Comment
Param0 Length – Param0 value Param1 Length – Param1 value Param2 Length – Param2 value Param3 Length – Param3 value Param4 Length – Param4 value Param5 Length – Param5 value Param6 Length – Param6 value	90	00	The response includes a 2-byte length field.
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is not defined
	64	02	Private Key is not defined
	6A	86	Incorrect P1P2
	6D	30	Javacard Exception
ParamA Length – ParamA value	90	00	OK

23.5.12 The SetKeyParameter EC256 Command

This command sets elliptic curve parameters, including public and private key parameters. The Admin PIN is required, except for Parameter 6 (Public Key), which accepts either the User or Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	PIN required
00	88	00 Parameter A	Key Index [0,3]	Length	Admin
		01 Parameter B		Length	Admin
		02 Parameter Field (Z/pZ)		Length	Admin
		03 Parameter G (generator)		Length	Admin
		04 Parameter K (cofactor)		Length	Admin
		05 Parameter R (order of G)		Length	Admin
		06 Parameter W (Public Key) The public key is checked against the private key and reset if an error occurs.		41	Admin
		07 Parameter S (Private Key) The private key MUST be initialized before setting the public key		20	Admin

Response

SW1	SW2	Comment
90	00	OK
63	80	PIN required
69	85	Bad index
64	01	Public Key is defined
64	02	Private Key is defined
6A	86	Incorrect P1P2
6D	40	Javacard Exception

23.5.13 The SetKeyParameter RSA2048 Command

This command sets the RSA public and private key parameters.
It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1 b7b6b5b4 = param b1b0= KeyIndex	P2 modulo MSB	P3	PIN required
00	8C	00 Modulo	0	Length= 255	Admin
		10 Public Exponent	0	Length= 5	Admin
		20 P	0	Length=128	Admin
		30 Q	0	Length=128	Admin
		40 DP1	0	Length=128	Admin
		50 DQ1	0	Length=128	Admin
		60 PQ	0	Length=128	Admin

Response

SW1	SW2	Comment
90	00	OK
63	80	PIN required
69	85	Bad index
64	01	Public Key is defined
64	02	Private Key is defined
6A	86	Incorrect P1P2
6D	40	Javacard Exception

23.5.14 The SignECDSA command

This command generates an ECDSA or PKCS#1 1.5 signature.

It requires the AdminPin or UserPin.

On success, it returns the status word SW1, SW2 = 9000.
Otherwise, it returns SW1=63, SW2=80 (PIN required).

Command

CLA	INS	P1	P2	P3	Data	PIN required
00	80	00 Raw Signature	b5: AlgoId 0=ECDSA, 1=PKCS#1 b1b0 Key Index FE for APP Key	Length 20	Data to be signed	Admin or User
00	80	21 Signature with SHA256	Key Index [0,3] FE or 5 for Key Cert Only for ECDSA	Length	Data to be hashed and signed	Admin or User

Response

Body	SW1	SW2	Comment
Length (2 bytes) ASN.1 ECDSA Signature Encoding Or RSA 256-byte PKCS#1 Signature	90	00	OK
	63	80	PIN required
	69	85	Bad index
	64	01	Public Key is not defined
	64	02	Private Key is not defined
	6A	86	Incorrect P1P2
	6D	20	Signature Error

23.5.15 The ECDHE command

This command performs a Diffie-Hellman key exchange over a 256-bit elliptic curve and optionally generates a public/private key pair. It requires no PIN.

Command

CLA	INS	P1	P2	P3	Data
00	8A	0	FF: generate an ephemeral public key over the SECP256 prime curve.	41	PublicKey
		0	0...3 key index	41	PublicKey

Response

Body	SW1	SW2	Comment
DH value, 32 bytes	90	00	OK
Empty	69	95	Conditions Not Satisfied
Empty	64	03	Private key not defined
Empty	64	04	Public key not defined

23.5.16 The HMAC Command

This command computes keys used by TLS 1.3.
It requires the Admin or User PIN.

Command

CLA	INS	P1	P2	P3	Data	PIN
00	85	00	02= Compute HMAC LengthKey [Key] LengthData [Data]	2+ LengthKey+ LengthData	32 bytes	Admin or User
		00 FF	0A= EXTRACT_EARLY P1=00 return ESK P1=FF return ESK, HSK, eBSK, rBSK, feBSK, frBSK	23	01 00 20 PSK (32 bytes)	Admin or User
		00 01	0B=EXPAND_EARLY P1=0 "c e traffic" P1=1 "e exp master"	3+ DataLength	00 20 DataLength Data	Admin or User
		00	0C= HMAC_EBSK	Data Length	Data	Admin or User
		00	0D=HMAC_RBSK	Data Length	Data	Admin or User
		00	0E=EXTRACT_HANDSHAKE (HMAC_HSK)	Data Length	Data (PubKey)	Admin or User
		00	01= ADD Mod N N as defined for SECP256k1	Data Length =32	Data	Admin or User
		00 FF xy	05= AESCCM Encrypt P1= 00 AD length=5 P1=FF no AD Other= AD Length	16 (KEY) 12 (IV) AD Length+ Data Length	AESKey IV AD Data	Admin or User
		00 FF xy	06= AESCCM Decrypt P1= 00 AD length=5 P1=FF no AD Other= AD Length	16 (KEY) 12 (IV) AD Length+ Data Length+ 16 (Tag)	AESKey IV AD Data Tag	Admin or User
		16	Compute SHA256 P1= number of SHA256 updates	Data Length	Data	Admin or User

Response

Body	SW1	SW2	Comment
P2=02, 0A, 0B, 0C, 0D, 0E, 16: 32 bytes 6x32 for EXTRACT_EARLY with P1=FF	90	00	No Error
P1= 01, 32 bytes	90	00	No Error
P1= 05 AD EncryptedData Tag	90	00	No Error
P1=06 Decrypted Data	90	00	No Error
Empty	6A	86	Wrong P1 P2

23.5.17 The GenerateRandom command

This command generates random bytes. It requires no PIN.

Command

CLA	INS	P1	P2	P3	PIN required
00	8B	00	00	Length	NO

Response

Body= P3 bytes or empty	SW1	SW2	Comment
	90	00	No Error
	69	85	Error

23.5.18 The GetCertificate command

This command reads the card certificate.

It requires the Admin or User PIN.

On success, it returns the status word SW1, SW2 = 9000.

Otherwise, it returns SW1=63, SW2=80 (PIN required).

The card certificate is an ECDSA SHA256 signature, over the secp56k1 curve, of the public key at index 0 (in full representation, 65 bytes). It consists of two concatenated 32-byte integers, r and s.

Command

CLA	INS	P1	P2	P3	PIN required
00	8E	00	00	00	Admin or User

Response

Body	SW1	SW2	Comment
64-byte certificate	90	00	Two 32-byte integers, r and s.
Empty	63	80	PIN required

23.5.19 The SetCertificate command

This command sets the card certificate.

It requires the Admin PIN.

On success, it returns the status word SW1, SW2 = 9000.

Two operating modes are supported: importing a 64-byte certificate or computing the certificate internally using an embedded key (self-signing).

Command

CLA	INS	P1	P2	P3	Payload	PIN required
00	8F	00	00	40	64-byte certificate	Admin
00	8F	FF	KeyIndex	00	Empty	Admin

Response

SW1	SW2	Comment
90	00	Certificate loaded or generated
63	80	PIN required
69	84	Certificate Already Set
69	85	Wrong Certificate Length
64	03	Private Key is not set

23.5.20 The SEND Command

This command writes a TLS packet to the secure element and returns a TLS message.

It requires no PIN.

Command

CLA	INS	P1	P2	P3	Data
00	D8	0 Normal Operation 1 if tls is open: Decrypt 2 if tls is open: Encrypt 3 if tls is open: Echo FF Rx Echo 1 in ClientHello PSK test 81 in ClientHelloPKI test	0:more 1: first 2: last 3: first&last	Data Length 00: Start/Reset TLS session	Data to be written

Response

Body	SW1	SW2	Comment
Optional message	90	00	OK
Optional message	90	01	TLS channel is open
Optional message	90	02	TLS channel is closed
Optional message	61	length	More data (length bytes) to read
Optional message	9F	length	More data (length bytes) to read
Empty	6D	02	Write Error
Empty	6D	01	Read Error
Empty	6D	14	Decryption Error
Empty	6D	15	Encryption Error
Empty	6D	16	TLS Error

23.5.21 The RECV Command

This command reads a TLS packet.
It requires no PIN.

Command

CLA	INS	P1	P2	P3
00	C0	0	0	Data Length

Response

Body	SW1	SW2	Comment
TLS Packet	90	00	No more data to read
TLS Packet	61	Length	More data (length bytes) to read
TLS Packet	9F	Length	More data (length bytes) to read

23.5.22 The TEST command

This command performs test operations.
It requires the Admin PIN.

Command

CLA	INS	P1	P2	P3	Data
00	C0	0	01 DH test	20	DiffieHellman
		0	02 Public Key test	41	PublicKey
		0	03 random test	20	Random value
		0	0A Server Name	length	ServerName
		0	FF Shell Call	length	Shell Command

Response

Body	SW1	SW2	Comment
Optional Body	90	00	OK
Empty	67	00	Wrong Length
Empty	6D	02	Write Error
Empty	63	80	PIN required
Empty	6A	86	Wrong P1 P2