EtherTrust CryptoTerminal
V1.01
Firmware 1.01h’



1 Content

2 INTRODUCTION 8

2.1 The Crypto Terminal 8

2.2 The Crypto Terminal Security Model 9

2.3 The Crypto Currency SmartCard (CCSC) 10

2.4 Generating Ethereum transaction from the terminal 10

2.5 Generating transactions from Android mobile 11

2.6 Generating transactions with BTOOLS 12

2.7 Overview of the ECTv2 crypto terminal board 13

3 UPLOADING FIRMWARE 13

3.1 USPASP Security procedure 13

3.2 Downloading the Crypto Terminal firmware 13

3.3 Downloading the Bluetooth Low Energy firmware 14

4 CHECKING FIRMWARE INTEGRITY 14

5 CHECKING SMARTCARD AUTHENTICITY 15

6 CHECKING CARD CONTENT 15

7 CHECKING MAIN PROCESSOR IDENTITY 16

8 KEYPAD 17

8.1 Root Menu (Iddle Menu) Keys 17

9 CRYPTO-TERMINAL SERIAL COMMANDS 18

9.1 Wrong Command 18

9.2 Empty Command 18

9.3 reboot 18

9.4 echo 18

9.5 disp 18

9.6 edit 19

9.7 enter 19
9.8 debug 19
9.9 nodebug 19
9.10 prompt 19
9.11 noprompt 19
9.12 on 20
9.13 off 20
9.14 status 20
9.15 changeadm 20
9.16 changeuser 20
9.17 changeuser2 21
9.18 adm 21
9.19 user 22
9.20 user2 22
9.21 computekey 22
9.22 genkey 23
9.23 setseed 23
9.24 getseed 24
9.25 getpub 24
9.26 btc 24
9.27 hash160 24
9.28 eth 24
9.29 clear 25
9.30 setkey 25
9.31 setpp 26
9.32 getpriv 26
9.33 setlabel 26
9.34 getlabel 27
9.35 sign 27

9.36 recover **27**

9.37 signr **27**

9.38 signin **28**

9.39 signout **28**

9.40 signoutr **28**

9.41 signdo **29**

9.42 settrans (generation of ether transaction) **29**

9.43 signt **30**

9.44 gettrans **30**

9.45 settransf (generation and signature of ethereum transaction) **31**

9.46 duplicate **31**

9.47 read **32**

9.48 write **32**

9.49 bMAC **32**

9.50 check **33**

9.51 tecc **33**

9.52 binder **33**

9.53 derive **33**

9.54 content **33**

10 CRYPTOTERMINAL MENUS **35**

10.1 Boot Message (BM) **35**

10.2 Idle Menu (IM) **35**

10.2.1 CheckCard Menu 35

10.2.2 Serial USB Mode 35

10.2.3 Bluetooth Mode 35

10.2.4 BLE Loader Mode 35

10.2.5 Card Content 35

10.2.6 Bluetooth Menu (BTM) 36

10.2.7 BMAC Menu (BMM) 36

10.2.8 Boot Menu (BM) 36

10.2.9 A Menu (AM) 36

10.3 Bluetooth Setting Menu (BSM) **36**

10.3.1 AT mode 36

10.3.2 BT Pin 36

10.3.3 BT Name 36

10.4 bMAC Run Menu (BRM) 36
10.4.1 Run bMAC 36
10.4.2 Run bMAC+EEPROM 36
10.4.3 Run bMAC BLE 37

10.5 Mode Menu (MM) 37
10.5.1 Mode ON 37
10.5.2 Mode OFF 37
10.5.3 Mode ADM 37
10.5.4 Mode User 37
10.5.5 Mode User2 37

10.6 Key Menu (KM) 38
10.6.1 Key Index 38
10.6.1.1 Sign 38
10.6.1.2 Genkey 38
10.6.1.3 Record Index 38
10.6.1.4 Clear Key 38
10.6.2 Key Menu2 (KM2) 38
10.6.2.1 CKey (Compute Key) 38
10.6.2.2 SetTree 39
10.6.2.2.1 SetTree 39
10.6.2.2.2 Read 39
10.6.2.2.3 GenerateTree 39
10.6.2.3 GetL (Get Label) 39
10.6.2.4 SetL (Set Label) 39
10.6.3 Key Menu3 (KM3) 40
10.6.3.1 SetPr (Set Private Key) 40
10.6.3.2 GetPr (Get Private Key) 40
10.6.3.3 GetPub (get public key) 40
10.6.3.3.1 Raw 40
10.6.3.3.2 BTC 40
10.6.3.3.3 Hash160 40
10.6.3.3.4 Ethereum Address 40
10.6.3.4 GetT (Get Tree Seed) 41

10.7 Duplicate Card Menu (Dup) 41
10.7.1 Remove Card 41
10.7.2 Insert New Card 41
10.7.3 Enter ADM PIN 41

10.8 Transaction Menu (Tr) 41
10.8.1 Ethereum Menu (EM) 41
10.8.1.1 Create transaction 41
10.8.1.1.1 Recipient Address 41
10.8.1.1.2 Ether Amount 41
10.8.1.1.3 Tx Data (text) 41
10.8.1.1.4 Transaction Nonce 41
10.8.1.1.5 Gas Price 41
10.8.1.1.6 Gas Limit 41
10.8.1.1.7 Tx Data (Hexadecimal) 41
10.8.1.2 Delete 41
10.8.1.3 Edit 42
10.8.1.3.1 Recipient Address 42
10.8.1.3.2 Ether Amount 42
10.8.1.3.3 Tx Data (Text) 42
10.8.1.3.4 Transaction Nonce 42

10.8.1.3.5 Gas Price ... 42
10.8.1.3.6 Gas Limit ... 42
10.8.1.3.7 Tx Data (Hexadecimal) ... 42
10.8.1.4 Sign ... 42

11 BTOOLS SCRIPT COMMANDS ... 43

11.1 Starting a serial terminal ... 43

11.2 CryptoTerminal Script ... 43
11.2.1 Index Array ... 43
11.2.2 start ... 43
11.2.3 on ... 43
11.2.4 off ... 44
11.2.5 adm ... 44
11.2.6 user ... 44
11.2.7 user2 ... 44
11.2.8 getpub ... 44
11.2.9 sign ... 44
11.2.10 signout ... 44
11.2.11 setpub ... 45
11.2.12 signdo ... 45
11.2.13 signin ... 45
11.2.14 raw ... 45

12 BTOOLS CRYPTOTERMINAL TRANSACTION SCRIPT EXAMPLES ... 46

12.1 Ethereum Transaction Generation ... 46
12.1.1 CryptoTerminal script ... 46
12.1.2 EthereumTransaction Script ... 46

12.2 Bitcoin Transaction Generation ... 46
12.2.1 CryptoTerminal script ... 46
12.2.2 Bitcoin Transaction Script ... 47

13 THE CRYPTO CURRENCY SMARTCARD ... 48

13.1 The Select Command ... 48

13.2 The Verify UserPin Command ... 48

13.3 The Verify UserPin2 Command ... 49

13.4 The Verify AdminPin command ... 49

13.5 The ChangePin command ... 50

13.6 The GetStatus command ... 50

13.7 The Write Command ... 51

13.8 The Read Command ... 51

13.9 The Clear KeyPair & InitCurve Command ... 52

13.10 The InitCurve & InitTree Command 53

13.11 The Generate KeyPair Command 54

13.12 The Dump KeyPair Command 55

13.13 The GetInfo command 57

13.14 The Get KeyParameter Command 58

13.15 The Set KeyParameter Command 59

13.16 The SignECDSA command 59

13.17 The GetCertificate command 60

13.18 The SetCertificate command 61

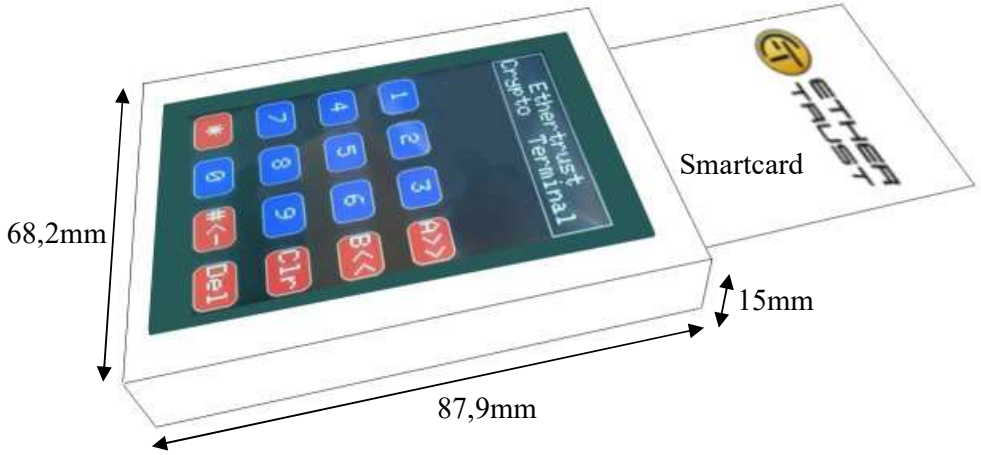
2 Introduction

Blockchain technology and its distributed ledger architecture have the potential to transform many industries. However, security, particularly the protection of cryptographic keys, remains a critical challenge, as demonstrated by past attacks.

Drawing on its extensive expertise in cryptography and secure-element technologies, EtherTrust brings proven payment-card security to blockchain transactions. With the EtherTrust Crypto Terminal, transactions can be authorized *offline* (*cold mode*) or *online* (*hot mode*) using c*hip-and-PIN* verification.

All key operations are performed within the smartcard secure element. For additional protection, the terminal firmware can be securely flashed immediately before a transaction.

2.1 The Crypto Terminal

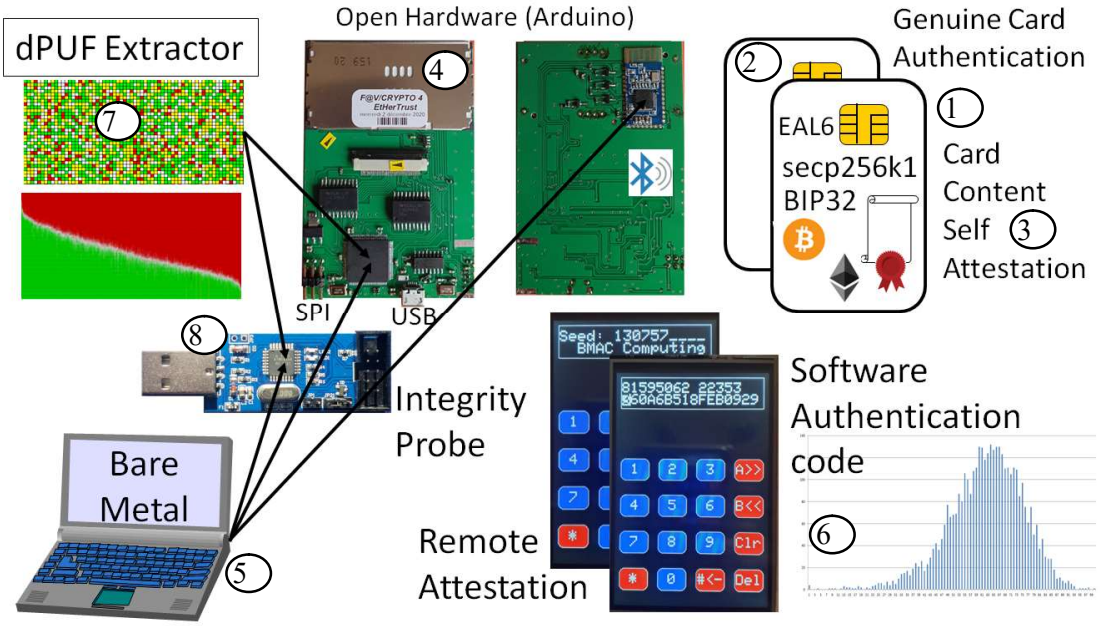


The Crypto Terminal has a USB port supporting the *USB serial* protocol and a *Bluetooth Low Energy* (BLE) module with an RFCOMM interface. It also includes a smartcard reader and provides the following functions:
- Generation and secure storage of cryptographic keys;
- Generation of cryptographic signatures in synchronous mode (controlled by a PC or mobile device) or asynchronous mode (without a connection to a PC or mobile device);
- Entry of all PINs directly on the Crypto Terminal, as with payment-card terminals;
- Confirmation of all signatures on the Crypto Terminal;
- High security:
    - *A removable secure element* (smartcard) with EAL5+/EAL6 certification;
    - *Bare-metal* operation, with Crypto Terminal and BLE-module firmware that can be flashed at any time.

The Crypto Terminal supports USB On-The-Go (OTG), which is available on most Android phones.

Power consumption is approximately 150 mA at 5 V (0.75 W). The dimensions are 87.9 × 68.2 × 15 mm, approximately the footprint of a payment card with a thickness of 15 mm.

2.2 The Crypto Terminal Security Model



1) The security core is a removable EAL5+/EAL6 secure element (Java Card) that stores cryptographic keys and signs transactions. Access to the smartcard is protected by PINs.

2) The Crypto Terminal can duplicate the contents of a secure element. The smartcard content is self-attested: it is hashed and signed by the smartcard.

3) Smartcard authenticity is verified through an anti-cloning mechanism designed to detect "evil maid" attacks, in which an attacker substitutes a cloned card to capture the user's keys. The smartcard public key is stored at index 0 and is signed by EtherTrust.

4) The Crypto Terminal acts as a firewall between the secure element and *online* environments connected to the Internet. It can also perform operations, including key generation and signing, in *cold* mode without an Internet connection.

5) The security model uses a "bare-metal" approach: the terminal and Bluetooth module are supplied without firmware. Users can flash both devices at any time using a programming token.

6) Firmware authenticity is checked using an embedded remote-attestation algorithm. The algorithm produces an authentication code that cannot be predicted in advance and whose computation time depends on the result. The EtherTrust server or an authorized user can generate these codes, allowing the use of unique personal authentication codes.

7) The physical authenticity of the Crypto Terminal processor and the programming-token processor is checked using dynamic SRAM p*hysical unclonable function* (dPUF) techniques. A memory-probing program extracts SRAM fingerprints after power-up. EtherTrust has developed techniques for capturing these fingerprints, including distinctive "flipping bits" that cannot reliably be replicated in software. This helps verify that devices supplied to customers are genuine.

8) The programming token acts as the root of trust and manages firmware loading onto the Crypto Terminal processor. EtherTrust installs a bootloader on the programming-token processor. The authenticity of this software is verified by a remote-attestation algorithm that generates authentication codes displayed through flashing LEDs. The token's physical *dPUF* identity also helps detect clones.

2.3 The Crypto Currency SmartCard (CCSC)

The Cryptocurrency Smartcard (CCSC) can generate, derive (in accordance with BIP32), or import up to 15 elliptic-curve keys for producing ECDSA signatures for cryptocurrencies such as Bitcoin and Ethereum.
The Cryptocurrency Smartcard (CCSC) application has AID *010203040500* and three PINs: administrator, user, and user2. The default administrator PIN is eight zeros; the default user and user2 PINs are four zeros each.
A 16 KB non-volatile read/write memory area, protected by a dedicated PIN (User2), is available for storing sensitive information.

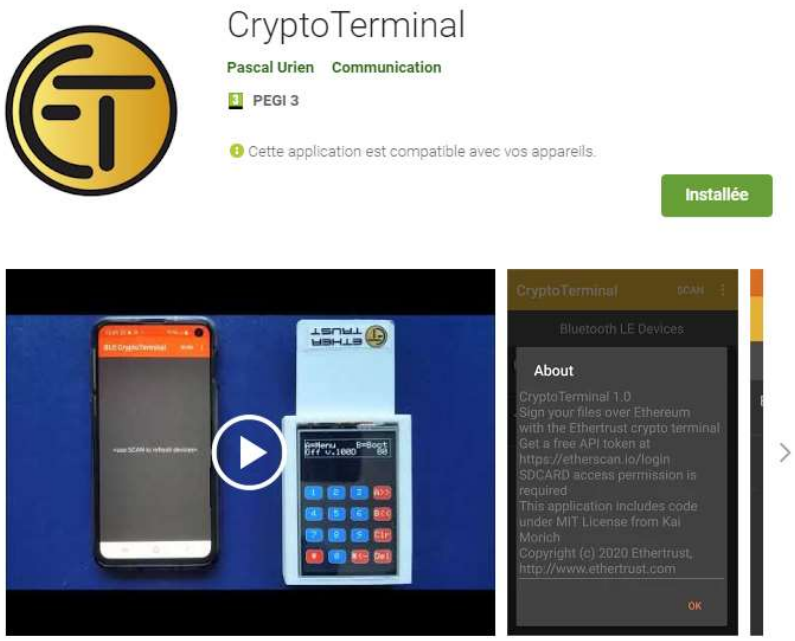
2.4 Generating an Ethereum Transaction on the Crypto Terminal

The serial command "*settransf*" generates an Ethereum transaction using the following seven parameters:
- Key index: the key used to sign the transaction;
- Nonce: a non-negative integer starting at zero and incremented for each transaction;
- GasPrice: the transaction gas price, in Gwei;
- GasLimit: the maximum gas allowed for transaction execution, resulting in a maximum gas cost of GasLimit × GasPrice;
- Recipient address: the recipient's Ethereum address (20 bytes, or 40 hexadecimal digits);
- Amount: the transaction amount in ether, expressed as a decimal number (for example, 0.0);
- Data: up to 448 ASCII characters (with a # prefix) or 224 bytes in hexadecimal notation (with a #$ prefix).

The "*settransf*" command returns the Ethereum transaction in hexadecimal format.

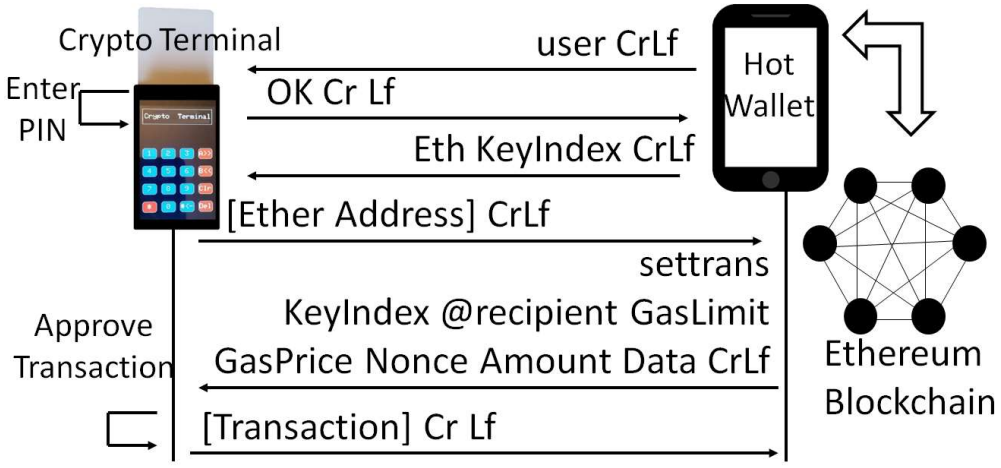
2.5 Generating Transactions from an Android Device

A CryptoTerminal application for Android is available on Google Play. The application *signs* a file stored on the mobile device by creating an Ethereum transaction.



https://play.google.com/store/apps/details?id=com.ethertrust.simple_terminal

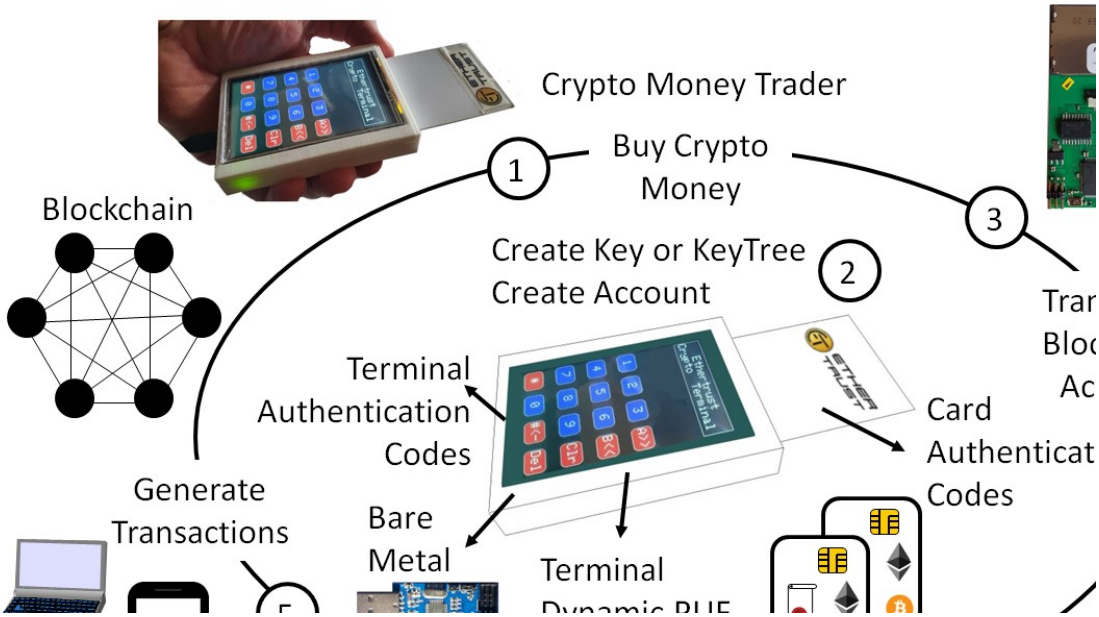
An RFCOMM serial connection is established between the Crypto Terminal and an Android phone. The *app*lication sends the "user" command, prompting the user to enter a PIN on the terminal and thereby unlock the smartcard. It then retrieves the Ethereum addresses associated with the cryptographic keys and requests transaction generation and signing using the "settransf" command. The user must approve the transaction.



2.6 Generating Transactions with BTOOLS

BTOOLS (https://github.com/purien/btools) is open-source software for Windows and Linux (versions 1.0 and 1.1 only) that generates Bitcoin and Ethereum transactions. For enhanced security, it supports the Crypto Terminal for private-key generation and storage and for computing elliptic-curve digital signatures (ECDSA). BTOOLS uses the OpenSSL library.

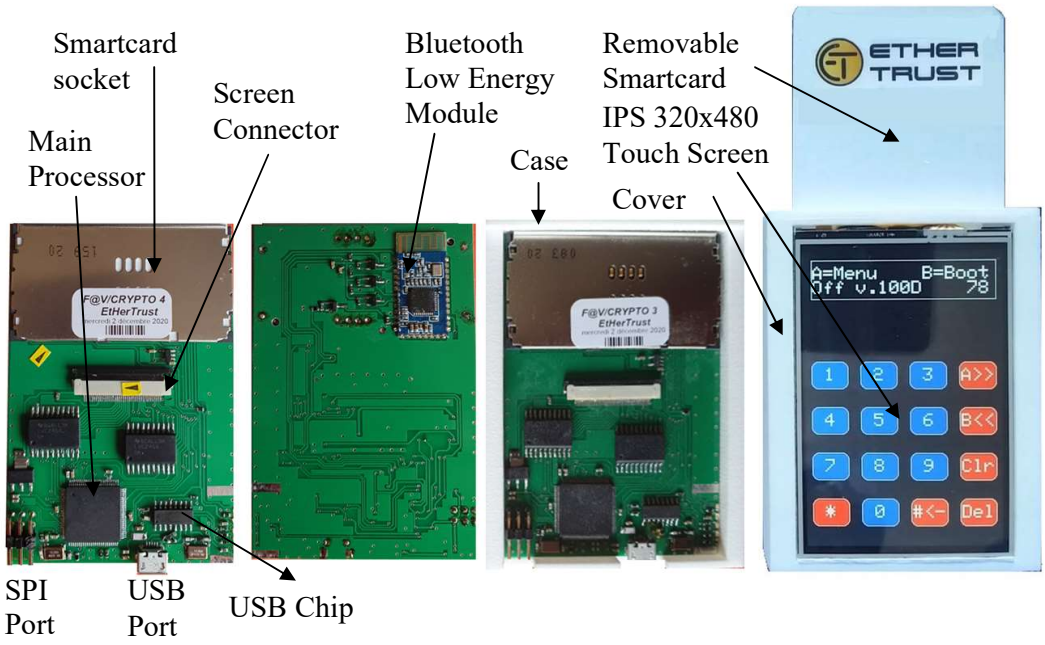
BTOOLS provides the following services:
- Bitcoin address generation (mainnet and testnet);
- Ethereum address generation;
- Bitcoin transaction generation;
- Ethereum transaction generation;
- Simple Bitcoin node client;
- Bitcoin transaction submission (through the Bitcoin client);
- Ethereum transaction submission (through web APIs);
- Crypto Terminal scripts for key generation and transaction signing (hot or cold mode).



The following workflow is recommended:

(1) Purchase cryptocurrency.
(2) Create a blockchain account (key or key tree) using the Crypto Terminal and smartcard.
(3) Transfer cryptocurrency to the blockchain account.
(4) Duplicate the smartcard.
(5) Generate transactions using software such as BTOOLS or a smartphone application.

2.7 Overview of the ECTv2 crypto terminal board



3 Uploading Firmware

The Crypto Terminal firmware can be loaded using any ISP programmer. The terminal has a six-pin ICSP port.

We recommend the open-source USBasp ISP programmer, for which EtherTrust has developed enhanced security features. The programmer includes an open bootloader customized by EtherTrust.
ICSP Port, Front View
and 6Pins/10Pins connector.

3.1 USBasp Security Procedure

- Short the RESET pin to ground.
- The green LED flashes three times, indicating that the bootloader is unlocked and ready to accept firmware for five seconds.
- Load the bootloader static-integrity probe firmware. If the test passes, the green and red LEDs flash alternately.
- Load the bootloader dynamic-integrity probe firmware. After approximately 20 seconds, the LEDs flash a five-digit security code. Verify the code.
- If both tests pass, load the USBasp firmware.

3.2 Downloading the Crypto Terminal firmware

Connect the six-pin ISP cable to the Crypto Terminal ICSP port. A six-pin-to-ten-pin adapter is required for the ISP programmer.

Load the CryptoTerminal.hex file. The processor is an ATmega2560.

3.3 Downloading the Bluetooth Low Energy firmware

Connect the Crypto Terminal to a PC using a USB cable.

Launch the BLE Loader application (CCLoader.exe) on the PC and select the COM port used by the Crypto Terminal.

When prompted by the BLE Loader, press key 3 on the Crypto Terminal to enter *BLE firmware* loading mode.

4 Checking Firmware Integrity



The integrity of the Crypto Terminal firmware and BLE-module firmware can be checked at any time.
On the main menu, press key 7.

Press 1 to check the Crypto Terminal firmware integrity. Press 2 to perform the same check while erasing the EEPROM.
Press 3 to check the BLE-module firmware.

Enter a *seed* value: an integer from 0 to 2,147,483,647 ($2^{31} - 1$).

The result contains three values:
- Computation time (in units of 4 µs for the Crypto Terminal or 64 µs for the BLE module), shown on the left of the first line; and a five-digit (16-bit) security code, shown on the right. The security code is calculated as the XOR of the MAC and computation time.
- The 32-byte MAC value in hexadecimal notation.

```
81586363 57820
4FE958A1BF7C8658
```

Crypto-Terminal Checking
(computing time about 326 seconds)

```
1165679 34846
EA2C02C71657BD04
```

BLE-Module Firmware Checking
(computing time about 70s)

We recommend seed 0 when checking the BLE module. With this seed, the BLE-module check runs a legacy SHA-256 procedure. In this case, the returned MAC should be:
EA2C02C71657BD04AE7527007D11FAF6436E76225AEA7A15806C5D55C94F881E
For the BLE module, the computation time may vary slightly (±256 µs); therefore, the security code may have several possible values.

In summary, check terminal firmware integrity as follows:
- Enter a seed for the Crypto Terminal integrity check and verify the security code.
- Enter seed 0 for the BLE-module integrity check and verify the MAC.

5 Checking Smartcard Authenticity



EtherTrust smartcards can be authenticated. Each card stores a certificate and private key, and its public key is stored at index 0.

First enter the smartcard user PIN (A/1/4/PIN#). Then press key 0.

Check Card	Reading Cert	Reading Pub
Challenge	Genuine Card	

The card authentication procedure consists of the following steps:
- Read the card certificate (EtherTrust's signature on the card public key).
- Read the card public key (index 0).
- Send a 32-byte challenge for signing with the card private key.
- Retrieve and verify the ECDSA signature.

If authentication succeeds, the *message "Genuine Card*" is displayed.

6 Checking Card Content



The EtherTrust smartcard can generate a self-attestation of its content.
Private keys and their associated labels are hashed and signed.

First enter the smartcard user PIN (A/1/4/PIN#). Then press key 4.

Reading Pub	Reading Content	Checking Content
1234567890ABCDEF Content		

The card-content verification procedure consists of the following steps:
- Read the card public key (index 0).
- Send a 32-byte challenge (r) and read the card-content hash.
- Verify the ECDSA signature: signature = ECDSA(SHA-256(card-content-hash || r)).

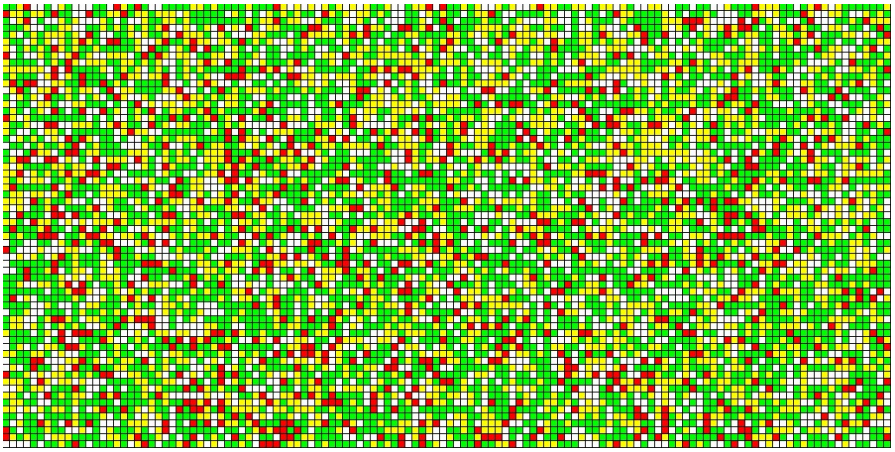
If verification succeeds, the card-content hash is displayed.

The EtherTrust public key (secp256k1) is:

046099836D971593AAA2C1C32B6DB9EF9521041795E21CF1E7511DF3BD358F97DF35
8B33A875E359CBE236163D6DBAEDFEC6C9393522C7EBC25A7CC85E1F0A7D67

7 Checking Main Processor Identity

The identity of the main processor is verified using a *dynamic physical unclonable function* (*d*PUF) procedure (see https://github.com/purien/DynamicPuf). First, memory-probe firmware is loaded onto the Crypto Terminal. A dedicated *PUF extractor* then measures or verifies the terminal's physical identity using its SRAM PUF. SRAM memory cells are built from six CMOS transistors.
Because of physical and electrical differences between transistors, some memory cells assume a repeatable value at power-up. Other SRAM cells, called "*flipping bits*," have values that depend predictably on the power-supply rise time. The processor's fingerprint consists of SRAM cells with stable power-up values (green = 1; yellow = 0) and flipping bits (red). Other cells are classified as noisy (white).



8 Keypad

The ten digit keys (0–9) are used to enter decimal digits.



The # key is used to confirm an entry.
The A key moves forward.
The B key moves backward.
The C key cancels an operation.
The D key deletes an entry.

The * key is used to escape.
To enter hexadecimal digits A through F, use the sequences *A, *B, *C, *D, *#, and **, respectively.

For telephone-keypad text entry, use the sequences *1x, *2x, *3x, and *4x, where x is the character index for the numeric key (1–9).

For ASCII characters, use *0xyz, where xyz is the ASCII code of the character.

For special characters, use *91 for a space, *92 for a period, and *93 for @.

For example, enter /A/2/2/0#/4 from the main menu to start the key-label editor.

8.1 Root Menu (Idle Menu) Keys

Key '0' : Check Card Authenticity
Key '1' : Serial USB activation
Key '2' : Bluetooth Low Energy activation
Key '3' : BLE-Module Loader activation
Key '4' : Card Content
Key '5' : Bluetooth options: PIN and Name
Key '7' : Firmware Integrity Check
Key '9' : Elliptic Curve Test
Key 'A' : Main Menu
Key 'B' : Boot Menu
Key 'D' : Product Name



9 Crypto-Terminal Serial Commands

The Crypto Terminal includes a USB port supporting the *USB serial* protocol and a *BLE m*odule with an *RFCOMM* stack.

- **Serial configuration: 9,600 baud, 8 data bits, 1 stop bit, no parity.**
- **All commands end with CR (carriage return) and LF (line feed) characters.**
- **An error is indicated when the first token of a response is ERROR.**

9.1 Wrong Command

Action: Command Error

Cmd: WrongCommand CrLf
The first token in an error response is **always** ERROR.
Response: ERROR WrongCommand CrLf

9.2 Empty Command

Action: Empty Command

Cmd: CrLf
Response: ERROR No Command ! CrLf

9.3 reboot

Action: Reboot the Crypto Terminal
Cmd: reboot CrLf
Response: Boot label
Boot label*: OK Ready EtherTrust Terminal v.101hL EEPROM Size: 4096 MaxRec* CrLf

9.4 echo

Action: Test the terminal; enable *nodebug* mode without changing the *prompt* setting.

Cmd: echo CrLf
Response: OK CrLf

9.5 disp

Action: Test the LCD for debugging; display text at x,y coordinates for five seconds.

Cmd: disp x-value (0..15) y-value (0...1) text CrLf
Response: OK CrLf
Response: ERROR CrLf

9.6 edit

Action: Test the LCD text editor for debugging.

Cmd: edit number-of-characters [Optional text]
Response: [text] CrLf
Response: ERROR CrLf

9.7 enter

Action: Test keypad input at x,y coordinates (60-second timeout).

Cmd: enter x-value (0..15) y-value (0...1) number-of-characters CrLf
Response: [text] CrLf
Response: ERROR CrLf

9.8 debug

Action: Switch to Debug Mode

Cmd: debug CrLf
Response: OK CrLf

9.9 nodebug

Action: Switch to normal (non-debug) mode.

Cmd: nodebug CrLf
Response: OK CrLf

9.10 prompt

Action: Display the prompt character (>) at the end of a response.

Cmd: prompt CrLf
Response: OK CrLf

9.11 noprompt

Action: Remove the prompt character (>) at the end of a response.

Cmd: noprompt CrLf
Response: OK CrLf

9.12 on

Action: Power the Secure Element

Cmd: on CrLf
Response: OK CrLf

9.13 off

Action: Unpower the Secure Element

Cmd: off CrLf
Response: OK CrLf

9.14 status

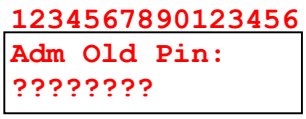
Action: read the Secure Element (SE) status parameter (10 bytes).
At least one PIN code must have been provided.

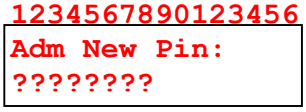
Cmd: status CrLf
Response: [SE Status Parameter, 10 bytes] CrLf
Response: ERROR CrLf

9.15 changeadm

Action: Modify Administrator PIN
This command powers the SE if needed.
Upon success, the terminal enters On mode.

Terminal Display:





Cmd: changeadm CrLf
Response: OK CrLf
Response: ERROR CrLf

9.16 changeuser

Action: Modify User PIN
This command powers the SE if needed.

Upon success, the terminal enters *On* mode.

Terminal Display:

```
1234567890123456
User Old Pin:
????
```

```
1234567890123456
User New Pin:
????
```

Cmd: changeuser CrLf
Response: OK CrLf
Response: ERROR CrLf

9.17 changeuser2

Action: Modify User2 PIN
This command powers the SE if needed.
Upon success, the terminal enters *On* mode.

Terminal Display:

```
1234567890123456
User2 Old Pin:
????
```

```
1234567890123456
User2 New Pin:
????
```

Cmd: changeuser2 CrLf
Response: OK CrLf
Response: ERROR CrLf

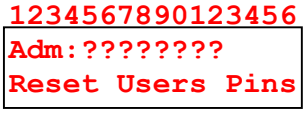
9.18 adm

Action: Enter the eight-digit administrator PIN on the Crypto Terminal and optionally reset the user1 and user2 PINs.
This command powers the SE if needed.

Terminal Display (default):

```
1234567890123456
Adm:????????
D=Del C=No  #=OK
```

Terminal Display (with the reset option):

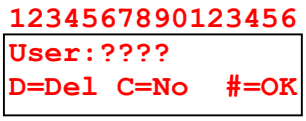


Cmd: adm [reset] CrLf
Response: OK CrLf
Response: ERROR CrLf

9.19 user

Action: Enter the four-digit user PIN on the Crypto Terminal.
This command powers the SE if needed.

Terminal Display:

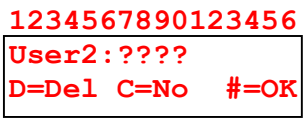


Cmd: user CrLf
Response: OK CrLf
Response: ERROR CrLf

9.20 user2

Action: Enter the four-digit user2 PIN on the Crypto Terminal.
This command powers the SE if needed.

Terminal Display:



Cmd: user2 CrLf
Response: OK CrLf
Response: ERROR CrLf

9.21 computekey

Action: Derive a private/public key pair from a key tree using hardened keys only, in accordance with BIP32.
Administrator or User mode is required; key generation must be confirmed.

Terminal Display:

```
1234567890123456
CKey iii: yes=#?
.x1.x2...xn time
```

Cmd: computekey iii x1.x2…xn CrLf
- iii an integer value between 0 and 15);
- xi integer value (between 0 and 2^{31}-1), the hardened key path in the key tree.

Response: OK CrLf
Response: ERROR CrLf

9.22 genkey

Action: Generate a private/public key pair.
Administrator mode is required; key generation must be confirmed.

Terminal Display:

```
1234567890123456
GenK iii: yes=#?
Adm          time
```

Cmd: genkey iii CrLf (iii an integer value between 0 and 15)
Response: OK CrLf
Response: ERROR CrLf

9.23 setseed

Action: Initialize a key tree with a supplied seed or generate a tree with a random seed of a specified length.
Administrator mode is required; key-tree initialization must be confirmed.

Terminal Display:

```
1234567890123456
Seed iii: yes=#?
Adm          time
```

Cmd: setseed iii seed CrLf (iii an integer value between 0 and 15)
Or
Cmd: setseed iii yy CrLf (iii an integer value between 0 and 15, yy integer the size of the random seed))

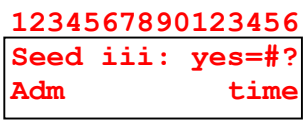
Response: OK CrLf
Response: ERROR CrLf

Seed: up to 128 hexadecimal digits (64 bytes)

9.24 getseed

Action: Read a key-tree seed.
Administrator mode is required; the read operation must be confirmed.

Terminal Display:



Cmd: getseed iii CrLf (iii an integer value between 0 and 15)
Response: [Length (2 bytes), Seed Value (up to 64 bytes) Ascii Encoded] CrLf
Response: ERROR CrLf

9.25 getpub

Action: read a public key.
Administrator or User PIN must have been entered.

Cmd: getpub iii CrLf (iii an integer value between 0 and 255)
Response: [Length (2 bytes), Public Key Value(65 bytes) Ascii Encoded] CrLf
Response: ERROR CrLf

9.26 btc

Action: compute a Bitcoin address.
Administrator or User PIN must have been entered.

Cmd: btc iii [xxx] CrLf (iii an integer value between 0 and 255, xxx the network ID an integer value between 0 and 255)
Response: [the Bitcoin address ascii encoded] CrLf
Response: ERROR CrLf

9.27 hash160

Action: compute a hash160.
Administrator or User PIN must have been entered.

Cmd: hash160 iii CrLf (iii an integer value between 0 and 255)
Response: [the hash160, 40 hexadecimal digits, ascii encoded] CrLf
Response: ERROR CrLf

9.28 eth

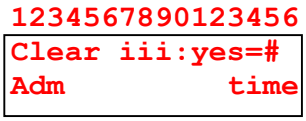
Action: Compute an Ethereum address.
Administrator or User PIN must have been entered.

Cmd: eth iii CrLf (iii an integer value between 0 and 255)
Response: [the Ether address, 40 hexadecimal digits, ascii encoded] CrLf
Response: ERROR CrLf

9.29 clear

Action: Clear a private/public key pair.
Administrator PIN must have been entered.

Terminal Display:

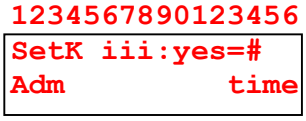


Cmd: clear iii CrLf (iii an integer value between 0 and 255)
Response: OK CrLf
Response: ERROR CrLf

9.30 setkey

Action: Set a private/public key pair.
Administrator PIN must have been entered.

Terminal Display:

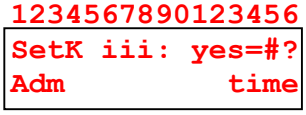


Cmd: setkey iii PrivKey PubKey CrLf (iii an integer value between 0 and 255)
Response: OK CrLf
Response: ERROR CrLf

PrivKey: 64 hexadecimal digits (32 bytes)
PubKey: 130 hexadecimal digits (65 bytes), in uncompressed point format.

Action: set private and public key.
Administrator PIN must have been entered.

Terminal Display:



9.31 setpp

Action: ***Set*** a private key and ***compute*** the corresponding public key.
Administrator PIN must have been entered.

Terminal Display:

```
1234567890123456
SetK iii: yes=#?
Adm          time
```

Cmd: setpp iii PrivKey CrLf (iii an integer value between 0 and 15)
Response: OK CrLf
Response: ERROR CrLf

PrivKey: 64 hexadecimal digits (32 bytes)

9.32 getpriv

Action: Read a private key.
Administrator PIN must have been entered.
The command must be confirmed.

Terminal Display:

```
1234567890123456
Priv iii: yes=#?
Adm          time
```

Cmd: getpriv iii CrLf (iii an integer value between 0 and 15)
Response: [Length (2bytes) Private Key(32 bytes), Ascii Encoded] CrLf
Response: ERROR CrLf

9.33 setlabel

Action: Set a key label.
Administrator PIN must have been entered.
The command must be confirmed.

Terminal Display:

```
1234567890123456
Label iii:yes=#?
>text
```

Cmd: setlabel KeyIndex (integer 0…15) "text" CrLf
Response:OK
Response: ERROR CrLf

9.34 getlabel

Action: Read a key label.
Administrator or User PIN must have been entered.

Cmd: getlabel KeyIndex (integer 0…15) CrLf
Response: [keylabel] OK
Response: ERROR CrLf

9.35 sign

Action: Generate a signature.
User or Administrator PIN must have been entered.
The command must be confirmed.

Terminal Display:

```
1234567890123456
Sign iii: yes=#?
User        time
```

Cmd: sign iii [hash value, Ascii Encoded] CrLf (iii an integer value between 0 and 15)
Response: [Length(2bytes), Signature (in ASN.1 syntax), Ascii Encoded] CrLf
Response: ERROR CrLf

9.36 recover

Compute and return the last recovery value (0 or 1).

Cmd: recover iii CrLf (iii an integer value between 0 and 15)
Response:[0 or 1] CrLf
Response: ERROR CrLf

9.37 signr

Action: Generate a signature and compute the recovery value.
User or Administrator PIN must have been entered.
The command must be confirmed.

Terminal Display:

```
1234567890123456
Sign iii: yes=#?
User        time
```

Cmd: signr iii [hash value, Ascii Encoded] CrLf (iii an integer value between 0 and 15)
Response: [Length(2bytes), Signature (in ASN.1 syntax), Ascii Encoded] RecoverValue (0/1) CrLf

Response: ERROR CrLf

9.38 signin

Action: Remote Signature Request.
The command must be confirmed.

Terminal Display:

```
1234567890123456
Sign iii: yes=#?
Off          time
```

Cmd: signin iii [hash value, Ascii Encoded] record CrLf
- iii an integer value between 0 and 255
- Hash value: the value to be signed (up to 32 bytes).
- Record: the record number (default = 0; integer from 0 to 20).
Response:OK CrLf
Response: ERROR CrLf

9.39 signout

Action: Read Remote Signature
The command must be confirmed.

Terminal Display:

```
1234567890123456
Sign iii: yes=#?
Off          time
```

Cmd: signout iii [HashValue, Ascii Encoded]
- iii an integer value between 0 and 15
- HashValue, the value to be signed
Response: : [Length (2bytes), Signature (in ASN.1 syntax) Hexa Ascii Encoded] [Length (2bytes), Public Key value uncompress format Hexa Ascii Encoded] CrLf
Response: ERROR CrLf

9.40 signoutr

Action: Read a remote signature and compute the recovery value.
The command must be confirmed.

Terminal Display:

```
1234567890123456
Sign iii: yes=#?
Off          time
```

Cmd: signoutr iii [HashValue, Ascii Encoded]
- iii an integer value between 0 and 15
- HashValue, the value to be signed
Response: : [Length (2bytes), Signature (in ASN.1 syntax) Hexa Ascii Encoded] [Length (2bytes), Public Key value uncompress format Hexa Ascii Encoded] RecoverValur (0/1) CrLf
Response: ERROR CrLf

9.41 signdo

Action: Do Remote Signature
User or Administrator PIN must have been entered
The command must be confirmed.

Terminal Display:

```
1234567890123456
Sign iii: yes=#?
User         time
```

Cmd: signdo iii [hash value, Ascii Encoded] record
- iii an integer value between 0 and 255
- hash value, the value to be signed
- record: the record number (0...20)

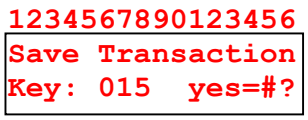
Response: : OK CrLf
Response: ERROR CrLf

9.42 settrans (generation of ether transaction)

Action:
The serial command "settrans" generates an ether transaction using the following seven parameters:
- Key Index, the key to be used for the transaction signature
- Nonce: a non-negative integer starting at zero and incremented for each transaction.
- GasPrice: the gas price in Gwei.
- GasLimit: the maximum gas allowed for the transaction, yielding a maximum gas cost of GasLimit × GasPrice.
- Recipient address: the recipient's Ethereum address (20 bytes, or 40 hexadecimal digits).
- Amount: the transaction amount in ether, expressed as a decimal number (for example, 0.0).
- Data: transaction data (text or hexadecimal, up to 448 characters).
    - Text example: "Hello World"
    - Text example: #Hello World
    - Hexadecimal example: $313233

The command must be confirmed.

Terminal Display



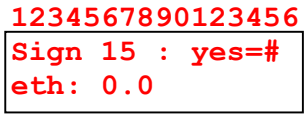
Cmd: settrans KeyIndex Nounce GasPrice GasLimit RecipientAddress Amount Data CrLf
Response: :OK CrLf
Response: ERROR CrLf

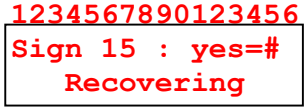
9.43 signt

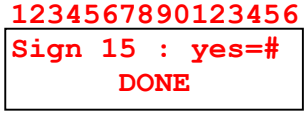
Action:
The "signt" command signs a previously created ether transaction.

The command must be confirmed.

Terminal Display





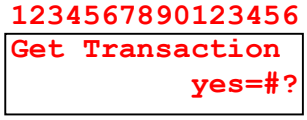


Cmd: signt
Response: :OK CrLf
Response: ERROR CrLf

9.44 gettrans

Action: Read a signed transaction.
The command must be confirmed.

Terminal Display



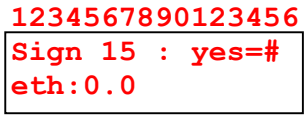
Cmd: gettrans CrLf
Response: : [Transaction encoded in hexadecimal] CrLf
Response: ERROR CrLf

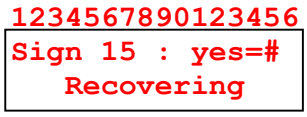
9.45 settransf (generation and signature of ethereum transaction)

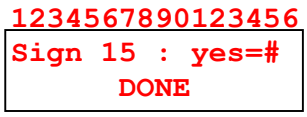
Action:
The "settransf" command generates and signs an Ethereum transaction using the following seven parameters:
- Key Index, the key to be used for the transaction signature
- Nonce: a non-negative integer starting at zero and incremented for each transaction.
- GasPrice: the gas price in Gwei.
- GasLimit: the maximum gas allowed for the transaction, yielding a maximum gas cost of GasLimit × GasPrice.
- Recipient address: the recipient's Ethereum address (20 bytes, or 40 hexadecimal digits).
- Amount: the transaction amount in ether, expressed as a decimal number (for example, 0.0).
- Data: transaction data (text or hexadecimal, up to 448 characters).
    - Text example: "Hello World"
    - Text example: #Hello World
    - Hexadecimal example: $313233

The command must be confirmed.

Terminal Display







Cmd: settrans KeyIndex Nounce GasPrice GasLimit RecipientAddress Amount Data CrLf
Response: : [Transaction encoded in hexadecimal] CrLf
Response: ERROR CrLf

9.46 duplicate

Action: Duplicate a CCSC smartcard onto another CCSC smartcard.
The administrator PIN must have been entered.
The command must be confirmed.

Cmd: duplicate CrLf
Response: :OK CrLf
Response: ERROR CrLf

Terminal Displays:

```
1234567890123456
Remove Card
#=Confirm ?
```

```
1234567890123456
Insert new card
#=Confirm ?
```

```
1234567890123456
Adm:????????
D=Del C=No  #=OK
```

9.47 read

Action: Read data from the smartcard's non-volatile user memory.
The user2 or administrator PIN must have been entered.

Cmd: read address size
address: data address (0...16383, decimal value), 16 KB available.
size: size of data to be read (1..256, decimal value)
Response: : [data (up to 256 bytes), hexadecimal ascii encoded] CrLf
Response: ERROR CrLf

9.48 write

Action: Write data to the smartcard's non-volatile user memory.
The user2 or administrator PIN must have been entered.

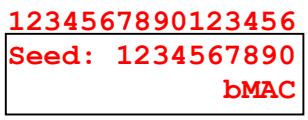
Cmd: write adr [data]
adr: data address (0...16383, decimal value), 16 KB available.
data: data to be written (1..255 bytes, hexadecimal ASCII-encoded)
Response: :OK CrLf
Response: ERROR CrLf

9.49 bMAC

Action: Check the Crypto Terminal firmware integrity.
Cmd: bmac seed [eeprom]
Seed: an integer from 0 to $2^{31}-1$.
EEPROM: an option that erases the EEPROM.

Response: :ComputingTime MAC CrLf
ComputingTime: computation time in units of 4 µs.
MAC: the 32-byte bMAC value.
Response: ERROR CrLf

Terminal Displays:



9.50 check

Action: Verify the authenticity of a CCSC smartcard.
The administrator or user PIN must have been entered.
The command must be confirmed.

Cmd: check CrLf
Response: :OK CrLf
Response: ERROR CrLf

9.51 tecc

Action: Test the ECC firmware.

Cmd: tecc CrLf
Response: :OK CrLf
Response: ERROR CrLf

9.52 binder

Action: Generate a cryptographic binder for TLS 1.3.
Administrator or User PIN must have been entered.

Cmd: binder 32bytes_hexadecimal_value CrLf
Response: 32bytes_hexadecimal_value CrLf
Response: ERROR CrLf

9.53 derive

Action: Compute the TLS 1.3 handshake secret.
Administrator or User PIN must have been entered.

Cmd: derive 32bytes_hexadecimal_value CrLf
Response: 32bytes_hexadecimal_value CrLf
Response: ERROR CrLf

9.54 content

Action: Read and verify the card content.

Administrator or User PIN must have been entered.

Cmd: content CrLf
Response: 32bytes_hexadecimal_value CrLf
Response: ERROR CrLf

10 CryptoTerminal Menus

The Crypto Terminal has a 320 × 480 IPS touchscreen that emulates a 16-key keypad and a 2 × 16-character LCD display.

10.1 Boot Message (BM)

```
1234567890123456
   EtherTrust
Crypto  Terminal
```

10.2 Idle Menu (IM)

Root Menu

```
1234567890123456
A=Menu     B=Boot
Mode         time
```

Mode: Off/On/Adm/User/User2/User1&2
Time: elapsed time in seconds since power-up.

10.2.1 CheckCard Menu

Path IM.0

```
1234567890123456
CHECK Card
```

```
1234567890123456
Reading Pub
```

```
1234567890123456
Reading Cert
```

```
1234567890123456
Challenge
```

```
1234567890123456
Genuine Card
```

10.2.2 Serial USB Mode

Path IM.1

```
1234567890123456
Serial USB
```

10.2.3 Bluetooth Mode

Path IM.2

```
1234567890123456
Bluetooth
```

10.2.4 BLE Loader Mode

Path IM.3

```
1234567890123456
BT Loader
```

10.2.5 Card Content

Path IM.4

```
1234567890123456
Reading Pub
```

```
1234567890123456
Reading Content
```

```
1234567890123456
Checking Content
```

```
1234567890123456
1234567890ABCDEF
         Content
```

10.2.6 Bluetooth Menu (BTM)

Path IM.5

```
1234567890123456
1=AT       AT=off
2=btPIN 3=btNAME
```

10.2.7 BMAC Menu (BMM)

Path IM.7

```
1234567890123456
1=bMAC 3=bMACble
2=bMAC+EEPROM  ?
```

10.2.8 Boot Menu (BM)

Path: IM.B

```
1234567890123456
Reboot(type B)=?
Mode         time
```

10.2.9 A Menu (AM)

Path: IM.A

```
1234567890123456
1=Mode    2=Key?
3=PIN 4=Dup 5=Tr
```

10.3 Bluetooth Setting Menu (BSM)

10.3.1 AT mode

Path: BTM.1

```
1234567890123456
Trying
OK
```

10.3.2 BT Pin

Path: BTM.2

```
1234567890123456
000000
btPIN
```

10.3.3 BT Name

Path: BTM.3

```
1234567890123456
ECTv2
btNAME
```

10.4 bMAC Run Menu (BRM)

10.4.1 Run bMAC

```
1234567890123456
Seed: 1234567890
#=OK  D=Del Time
```

```
1234567890123456
Seed: 1234567890
  BMAC Computing
```

```
1234567890123456
81586363 57820
4FE958A1BF7C8658
```

10.4.2 Run bMAC+EEPROM

```
1234567890123456
Seed: 130757
#=OK  D=Del Time
```

```
1234567890123456
Seed: 130757
     BMAC+EEPROM
```

```
1234567890123456
81776880 08341
F11E62C62DAD1793
```

10.4.3 Run bMAC BLE

```
1234567890123456
Seed: 0
#=OK  D=Del Time
```

```
1234567890123456
Seed: 1234567890
   BLE Computing
```

```
1234567890123456
1165679 34846
EA2C02C71657BD04
```

10.5 Mode Menu (MM)

Path: AM.1

```
1234567890123456
1=ON 2=OFF 3=ADM
4=USER 5=USER2 ?
```

10.5.1 Mode ON

Path MM.1

```
1234567890123456
  Starting SE
?              OK
```

10.5.2 Mode OFF

Path MM.2

```
1234567890123456
  Stopping SE
?              OK
```

10.5.3 Mode ADM

Path: MM.3

```
1234567890123456
  Starting SE
```

```
1234567890123456
Adm:????????
D=Del C=No  #=OK
```

```
1234567890123456
                Adm
?               OK
```

```
1234567890123456
                Adm
?             ERROR
```

10.5.4 Mode User

Path: MM.4

```
1234567890123456
  Starting SE
```

```
1234567890123456
User:????
D=Del C=No  #=OK
```

```
1234567890123456
               User
?               OK
```

```
1234567890123456
               User
?             ERROR
```

10.5.5 Mode User2

Path: MM.5

```
1234567890123456
  Starting SE
```

```
1234567890123456
User2:????
D=Del C=No  #=OK
```

```
1234567890123456
               User2
?               OK
```

```
1234567890123456
            User2
?           ERROR
```

10.6 Key Menu (KM)

Path AM.2

10.6.1 Key Index

```
1234567890123456
KeyIndex: ???
#=OK D=Del  C=No
```

```
1234567890123456
1=Sign 2=Gen 4=C
Key:iii 3=Rec A>
```

10.6.1.1 Sign

Path: KM.1

```
1234567890123456
Sign iii: yes=#?
Record: yy
```

```
1234567890123456
    sign iii rec
?             OK
```

```
1234567890123456
    sign iii rec
?          ERROR
```

10.6.1.2 Genkey

Path: KM.2

```
1234567890123456
GenK iii: yes=#?
#=OK D=Del  C=No
```

```
1234567890123456
      genkey iii
?             OK
```

```
1234567890123456
      genkey iii
?          ERROR
```

10.6.1.3 Record Index

Path KM.3

```
1234567890123456
RecordIndex: ??
#=OK D=Del  C=No
```

10.6.1.4 Clear Key

Path KM.4

```
1234567890123456
Clear iii yes=#?
```

```
1234567890123456
       clear iii
?             OK
```

10.6.2 Key Menu2 (KM2)

Path: KM.A

```
1234567890123456
1=CKey 2=SetTree
3=GetL 4=SetL A>
```

10.6.2.1 CKey (Compute Key)

Path: KM.2.1

```
1234567890123456
1234.10---------
 ComputeKey: dec
```

```
1234567890123456
CKEY iii: yes=#?
.1000.10
```

```
1234567890123456
ComputeKey
DONE           ?
```

```
1234567890123456
ComputeKey
ERROR              ?
```

10.6.2.2 SetTree

Path: KM2.2

```
1234567890123456
1=SetTree 2=Read
3=Generate Tree
```

10.6.2.2.1 SetTree

Path: KM2.2.1

```
1234567890123456
1234_____________
   SetSeed: hexa
```

```
1234567890123456
Seed iii: yes=#?
Col:005 n:004 U
```

```
1234567890123456
SetSeed
DONE               ?
```

```
1234567890123456
SetSeed
ERROR              ?
```

10.6.2.2.2 Read

Path: KM2.2.2

```
1234567890123456
Seed iii: yes=#?
Getseed 3          ?
```

```
1234567890123456
GetTreeSeed
ERROR              ?
```

```
1234567890123456
0123456789ABCDEF
  TreeSeed: hexa
```

10.6.2.2.3 GenerateTree

Path: Path: KM2.2.2

```
1234567890123456
Length 32 number
#=OK D=Del  C=No
```

```
1234567890123456
Seed iii: yes=#?
Col:005 n:004 U
```

```
1234567890123456
SetSeed
DONE               ?
```

```
1234567890123456
SetSeed
ERROR              ?
```

10.6.2.3 GetL (Get Label)

Path: KM2.3

```
1234567890123456
Hello World !
                   ?
```

```
1234567890123456
GetLabel
ERROR              ?
```

10.6.2.4 SetL (Set Label)

Path: KM2.4

```
1234567890123456
Hello World !
Setlabel 3
```

```
1234567890123456
Label iii:yes=#?
>Hello World!
```

```
1234567890123456
SetLabel
DONE               ?
```

```
1234567890123456
SetLabel
ERROR             ?
```

10.6.3 Key Menu3 (KM3)

Path: KM2.5

```
1234567890123456
1=SetPr   2=GetPr
3=GetPub  4=GetT
```

10.6.3.1 SetPr (Set Private Key)

Path KM3.1

```
1234567890123456
0123456789ABCDEF
SetPrivKey: hexa
```

```
1234567890123456
SetK iii: yes=#?
```

```
1234567890123456
SetPrivKey
DONE              ?
```

```
1234567890123456
SetPrivKey
ERROR             ?
```

10.6.3.2 GetPr (Get Private Key)

Path: KM3.2

```
1234567890123456
GetPrivKey
ERROR             ?
```

```
1234567890123456
Priv iii: yes=#?
getpriv 3
```

```
1234567890123456
0123456789ABCDEF
   PrivKey: hexa
```

10.6.3.3 GetPub (get public key)

Path KM3.3

```
1234567890123456
1=Raw       2=BTV
3=H160      4=ETH
```

10.6.3.3.1 Raw

Path KM3.3.1

```
1234567890123456
GetPubKey
ERROR             ?
```

```
1234567890123456
0437A4AEF1F8423C
    PubKey: hexa
```

10.6.3.3.2 BTC

Path KM3.3.1

```
1234567890123456
ID:0??    integer
#=OK  D=Del C=No
```

```
1234567890123456
1Mf9gx8gF9ocM42b
BTC Address 000
```

10.6.3.3.3 Hash160

Path KM3.3.2

```
1234567890123456
1234567890ABCDEF
 HASH160 Address
```

10.6.3.3.4 Ethereum Address

Path KM3.3.3

```
1234567890123456
1234567890ABCDEF
   ETHER Address
```

10.6.3.4 GetT (Get Tree Seed)

Path KM3.4

```
1234567890123456
Seed iii: yes=#?
Getseed 3       ?
```

```
1234567890123456
GetTreeSeed
ERROR           ?
```

```
1234567890123456
0123456789ABCDEF
  TreeSeed: hexa
```

10.7 Duplicate Card Menu (Dup)

Path AM.4

10.7.1 Remove Card

```
1234567890123456
Remove Card
#=Confirm ?
```

10.7.2 Insert New Card

```
1234567890123456
Insert new card
#=Confirm ?
```

10.7.3 Enter ADM PIN

```
1234567890123456
Adm:????????
D=Del C=No  #=OK
```

10.8 Transaction Menu (Tr)

Path AM.5

```
1234567890123456
KeyIndex: ???
#=OK D=Del C=No?
```

```
1234567890123456
Transaction  015
Eth=1           ?
```

10.8.1 Ethereum Menu (EM)

Path AM.5.1

```
1234567890123456
1=Create   2=Del
3=Edit    4=Sig?
```

10.8.1.1 Create transaction

Path EM.1

```
1234567890123456
1=@ 2=Eth 37=Dat
4=N. 5=Gas 6=Lim
```

```
1234567890123456
Save Transaction
          yes=#?
```

10.8.1.1.1 Recipient Address
Path EM.1.1

10.8.1.1.2 Ether Amount
Path EM.1.2

10.8.1.1.3 Tx Data (text)
Path EM.1.3

10.8.1.1.4 Transaction Nonce
Path EM.1.4

10.8.1.1.5 Gas Price
Path EM.1.5

10.8.1.1.6 Gas Limit
Path EM.1.6

10.8.1.1.7 Tx Data (Hexadecimal)
Path EM.1.7

10.8.1.2 Delete

Path EM.2

```
1234567890123456
Transaction
Delete     #=yes?
```

10.8.1.3 Edit

Path EM.3

```
1234567890123456
1=@ 2=Eth 37=Dat
4=N. 5=Gas 6=Lim
```

```
1234567890123456
Save Transaction
          yes=#?
```

10.8.1.3.1 Recipient Address
Path EM.3.1

10.8.1.3.2 Ether Amount
Path EM.3.2

10.8.1.3.3 Tx Data (Text)
Path EM.3.3

10.8.1.3.4 Transaction Nonce
Path EM.3.4

10.8.1.3.5 Gas Price
Path EM.3.5

10.8.1.3.6 Gas Limit
PATH EM.3.6

10.8.1.3.7 Tx Data (Hexadecimal)
Path EM.3.7

10.8.1.4 Sign

Path EM.4

```
1234567890123456
Sign 015: yes=#?
      DONE
```

11 BTOOLS Script commands

BTOOLS (2.0) is open software (*https://github.com/purien/btools*) available for WINDOWS and LINUX and RASPBERRYPI, which generates transactions for Bitcoin and Ethereum crypto currency platforms. It is based on the OPENSSL library. In order to provide a strong security, it supports it the Crypto Currency SmartCard (CCSC) uses to generate and store secret keys, and to compute cryptographic (ECDSA) signature.

BTOOLS (3.0) is compatible with the CryptoTerminal.

It supports two facilities, a serial terminal for manual interactions and scripts for programmed interactions.

11.1 Starting a serial terminal

A serial terminal is started by the command line:

btools –terminal delay.

The typical value for delay is 2000 (ms)

11.2 CryptoTerminal Script

A *CryptoTerminal* script has a name that begins by the "_" character.

It is a set of command lines ending by the CrLF characters.

11.2.1 Index Array

An integer value can be stored in an *Index Array* (IA[256]), whose index is %xy, where xy is an hexadecimal two digit integer. For example %A0 (160 in decimal) means that index = TA[160].
An index affectation in done in the transaction script, for example %A0 4, means that KA[160].= 4

11.2.2 start

Action: start a session with the CryptoTerminal

Cmd: start [optional COM port number] [optional delay in ms]

11.2.3 on

Action: power the CryptoTerminal secure element

Cmd: on

11.2.4 off

Action: unpower the CryptoTerminal secure element

Cmd: off

11.2.5 adm

Action: set the CryptoTerminal in Administrator mode

Cmd: adm

11.2.6 user

Action: set the CryptoTerminal in User mode

Cmd: adm

11.2.7 user2

Action: set the CryptoTerminal in User2 mode

Cmd: user2

11.2.8 getpub

Action: get the public key.

Cmd: getpub KeyIndex
- KeyIndex is a decimal value (0...255)

11.2.9 sign

Action: compute an ECDSA signature.

Cmd: sign KeyIndex
- KeyIndex is a decimal value (0...255)

11.2.10 signout

Action: export a ValueToSign for offline signature

Cmd: sign KeyIndex RecordIndex
- KeyIndex: a decimal value (0...255)
- RecordIndex: the record index to be used (0..20)

11.2.11 setpub

Action: set the public key value needed for offline signature
This command is used only for bitcoin family transactions, before signout and signin commands.

Cmd: setpub PublicKeyValue

11.2.12 signdo

Action: compute the offline signature

Cmd: signdo KeyIndex RecordIndex
- KeyIndex: a decimal value (0...255)
- RecordIndex: the record index to be used (0..20)

11.2.13 signin

Action: read an offline signature

Cmd: signin KeyIndex RecordIndex
- KeyIndex: a decimal value (0...255)
- RecordIndex: the record index to be used (0..20)

11.2.14 raw

Action: send a command to the CryptoTerminal

Cmd: raw [CryptoTerminal Command]

Examples:
raw setseed TreeIndex SeedValue
raw compute TreeIndex KeyPath

12 BTOOLS CryptoTerminal Transaction Script Examples

12.1 Ethereum Transaction Generation

btools -genethtrans escript.txt ecsript.bin

12.1.1 CryptoTerminal script

```
//////////////////
// _term.txt    //
//////////////////
start 2 1000
user
getpub %A0
sign   %A0
off
```

12.1.2 EthereumTransaction Script

```
////////////////////////
//    escript.txt     //
////////////////////////
%A0 5
apdu_script _term.txt
nonce     14
gasPrice  30000000000
gasLimit  80000
to         6BAC1B75185D9051AF740AB909F81C71BBB221A6
value      0
data       "Hello World"
// dataf   file.txt
```

12.2 Bitcoin Transaction Generation

btools -gentrans bscript.txt bscript.bin

12.2.1 CryptoTerminal script

```
//////////////////
// _term.txt    //
//////////////////
start 2 1000
user
getpub %A0
sign   %A0
off
```

12.2.2 Bitcoin Transaction Script

```
//////////////////
// bscript.txt //
//////////////////

sequence ffffffff
locktime 00000000

%A0 4

nb_input 1

input
transaction     3573f3007a83db8b0eb04a808bf76983dba8270be9c2df44a196
                97517ab2c062
index 1
apdu_script _term.txt

nb_output 2

output "Hello Ethertrust Crypto Terminal !"
output
fee 0.0010
btc 0.5270
adr mtekQi1EPrxo5TngwipB31VJTihSCTjxjk
```

13 The Crypto Currency SmartCard

The Crypto Currency smartcard (CCSC), of which AID is 010203040500 has three PINs, administrator, user, and user2. The default values are 8 zeros (3030303030303030) for administrator and 4 zeros (30303030) for user and user2.

It is able to generate, to compute according to the BIP32 standard, or to import elliptic curve keys (up to 16), used for the generation of ECDSA signatures used by Bitcoin and Ethereum crypto currencies.

A Read/Write non-volatile memory (16KB), protected by a dedicated PIN (User2), is available for the storage of any sensitive information.

13.1 The Select Command

This command starts the Crypto Currency smartcard application
Upon 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

13.2 The Verify UserPin Command

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

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

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

13.3 The Verify UserPin2 Command

This command verifies the second user pin. The UserPin2 is required for the memory reading and writing operations.
Upon success it returns the status word SW1 SW2 = 9000
Otherwise it returns SW1=63, SW2=number of remaining tries (3 at the most)

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

13.4 The Verify AdminPin command

This command verifies the administrator pin. It gives access to all available features of the crypto currency application. If P2 is set to FF UserPin and UserPin2 are reset to the default value (four zeros).

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

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

13.5 The ChangePin command

This command sets a PIN (UserPin, UserPin2, AdminPin) to a new value
The P2 value is respectively 00, 02, 01 for UserPin, UserPin2, AdminPin.
Upon 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

13.6 The GetStatus command

This command returns the current state of the crypto currency application. It required at least the previous checking of one PIN (UserPin, UserPin2, AdminPin).

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= BigNumber (OK= 0x07)
byte1: The maximum number of keys that can be used 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 the index (bi) of defined keys, for example 0003 for key1 (bit1) and key0 (bit0)
byte8, byte9: 16 bits (b15...b1b0) indicating the index (bi) of defined key trees, for example 0003 for tree1 (bit1) and tree0 (bit0)

13.7 The Write Command

This command writes data in the non-volatile memory. This service could be used for the secure storage of any information in the area [0000, 0C00[.

It requires the previous checking of PINs UserPin2 or AdminPin.
Upon 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 by two bytes (P1, P2), P1 being the *most significant byte* (MSB) and P2 the *less 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

13.8 The Read Command

This command reads data in the non-volatile memory.
It requires the previous checking of PINs UserPin2 or AdminPin.
Upon 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 by two bytes (P1, P2), P1 being the most significant byte (MSB) and P2 the less 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
Empy	6D	01	Invalid Address

13.9 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 EllipticCurve, excepted when the InitCurve option is used.

It requires the Admin PIN, or for P1=10 (Reset Key Tree) User or Admin PIN.

The P1=80 option is used to clear either the public or the private, and to initialize the associated curve. Here are some 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.

Upon 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			
00	81	SECP256k1 80 – ClearKeys & 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

13.10 The InitCurve & InitTree Command

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

The KeyTree is initialized by a seed according to the BIP32 specification. Two mode of seed generation 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.

Upon 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

13.11 The Generate KeyPair Command

This command generates the elliptic curve public and private keys or computes a key according to the BIP32 specification

The keys MUST be cleared before this operation.

It requires the Admin PIN or for KeyTree (P2 is a KeyTree index and P3#0) User PIN or Admin PIN

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

If P3 is a multiple of 4, a hardened private key is generated whose index a list of n 32 bits word, $IH_1/IH_2/\ldots/IH_n$. The public key is not computed. The MSB bit of IHi 32bits word MUST be set.

Upon 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

13.12 The Dump KeyPair Command

This command dumps the elliptic curve public and private keys.
It returns the size of the data written in the non-volatile memory, in the *Key Dump* area whose address starts at 0C00

It requires the Admin PIN.
Upon 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 P1P2

Dump KeysPair: 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

13.13 The GetInfo command

This command collects a list of information for a given key index, including key label, tree seed and private key.
It requires the Admin PIN, or the user PIN for the *CardContent* option.
Upon 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 byte) 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

13.14 The Get KeyParameter Command

This command collects the elliptic curve public and private keys parameters.

It requires the User or the Admin PIN.
Upon 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	Response includes a 2 bytes 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

13.15 The Set KeyParameter Command

This command sets the elliptic curve parameters, including public and private keys parameters It requires the Admin PIN excepted for Parameter 6 (Public Key) for which User or Admin PIN is required.

Upon 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 according to the private key, and is reset in case of error		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

13.16 The SignECDSA command

This command generates an ECDSA signature.

It requires the AdminPin or UserPin.

Upon 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

13.17 The GetCertificate command

This command reads the card certificate.
It requires the Admin or the User PIN.
Upon success it returns the status word SW1, SW2 = 9000.
Otherwise it returns SW1=63, SW2=80 (PIN required).
The card certificate is the ECDSA SHA256 signature over the secp56k1 curve, of its public key (in full representation, i.e. 65 bytes) located at index 0. This certificate is the concatenation of two 32 bytes integers value r, 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 bytes certificate	90	00	Two 32 bytes integers r,s
Empty	63	80	PIN required

13.18 The SetCertificate command

This command sets the card certificate.
It requires the Admin PIN.
Upon success it returns the status word SW1, SW2 = 9000.

Two working mode are supported. The certificate (64 bytes) is imported or the certificate is internally computed from an embedded key (auto signing)

Command

CLA	INS	P1	P2	P3	Payload	PIN required
00	8F	00	00	40	64 bytes 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