SECURITY ARTS

QUANTUM Communication Protocol

Protocol Version 1.0

Revision history

Date	Revision	Description	Author
2019-11-20	-	Initial version	Vasyl Parovinchak

Contents

1 Introduction

This document provides Security Arts QUANTUM communication protocol specification and API to help developers integrate QUANTUM to third party services.

2 Overview

The protocol consists of a set of commands that a client can perform on a server, where QUANTUM acts as a server. Communication is always initiated by a client in a command-response manner. Only one command could be executed in one moment of time.

The protocol consists of 3 logical layers:

- Transaction layer – highest layer. Each transaction consists of a command initiated by the client and the response sent by the server. All transactions are atomic - response should be received before next command could be sent.
- Packet layer – each command/response consists of one or more packets (64 bytes each). Each packet consists of 5 or 7 bytes of packet layer data and 59 or 57 bytes of payload (transaction layer data).
- Transport layer – USB HID is used as a transport layer to deliver packets between the client and the server.

3 Transport layer

QUANTUM supports full-speed USB 2.0 (12Mbits/s) communication.

VID/PID: 0x1A92/0x2013 – for USB HID.

3.1 USB interface and endpoints descriptors

USB device descriptor

Device descriptor	
bcdUSB	2.0
bDeviceClass	Class defined at interface level
bDeviceProtocol	None
bMaxPacketSize0	64
idVendor	0x1A92
idProduct	0x2013
bcdDevice	2.0
iManufacturer	1 "SecurityArts"
iProduct	2 "Quantum"
iSerialNumber	3 "[redacted]"

USB interface descriptor

Interface descriptor	
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	2
bInterfaceClass	Human Interface Device (Find out more online)

QUANTUM implements 2 (one IN and one OUT) endpoints for USB communication with 64 bytes packet size:

Endpoint descriptor	
bEndpointAddress	1 IN
bmAttributes. TransferType	Interrupt
wMaxPacketSize	64 bytes
bInterval	2 frames (2 ms)

Endpoint descriptor	
bEndpointAddress	1 OUT
bmAttributes. TransferType	Interrupt
wMaxPacketSize	64 bytes
bInterval	2 frames (2 ms)

3.2 USB HID report descriptor

QUANTUM implements two "raw" reports, which basically map directly to the IN and OUT endpoints.

```
const BYTE HID_ReportDescriptor[USB_HID_HID_REPORT_DESC_SIZE] =
{
    0x06, 0x00, 0xFF, // Usage Page (Vendor)
    0x09, 0x01,       // Usage (Vendor)
    0xA1, 0x01,       // Collection (Application), HID_APPLICATION
    0x09, 0x20,       //  Usage (Input Report Data), DATA IN
    0x15, 0x00,       //  Logical Minimum (0)
    0x26, 0xFF, 0x00, //  Logical Maximum (255)
    0x75, 0x08,       //  Report Size (8)
    0x95, 0x40,       //  Report Count (64), HID_INPUT_REPORT_BYTES
    0x81, 0x02,       //  Input (Data, Var, Abs), Usage
    0x09, 0x21,       //  Usage (Output Report Data), DATA_OUT
    0x15, 0x00,       //  Logical Minimum (0)
    0x26, 0xFF, 0x00, //  Logical Maximum (255)
    0x75, 0x08,       //  Report Size (8)
    0x95, 0x40,       //  Report Count (64), HID_OUTPUT_REPORT_BYTES
    0x91, 0x02,       //  Output (Data, Var, Abs), Usage
    0xC0              // End Collection
};
```

This way driver-less communication could be implemented in almost every modern operating system that supports the USB HID (Windows, Linux, Mac OS, Android and others).

3.3 Device discovery

Device discovery could be done using corresponding VID/PID. If more than one QUANTUM instance is connected to PC, an additional discovery procedure could be implemented using the serial number from the USB device descriptor.

The serial number is 24 symbols hex string and is unique for every QUANTUM device.

3.4 Recommendations

Security Arts QuantumManager is an open source multiplatform desktop application for the QUANTUM device and could be used as a reference.

It is developed using the Electron JS framework and uses node-hid library for USB communication.

usbhid.js is JS code that implements all three layers (Transaction, packet and transport) required to communicate with QUANTUM.

4 Packet layer

The packet layer is a logical “channel” that is used to split long transactions into 64 bytes packets and deliver them in the correct order.

4.1 Packet types

There are two types of packets: initialization and continuation packets.

Each transaction starts with one initialization packet followed by continuation packets.

If the transaction is 57 bytes long or less - it fits into one initialization packet and no more continuation packets are required.

Initialization packet format:

Offset	Size (bytes)	Name	Description
0	4	CID	Channel ID
4	1	CMD	Command
5	1	SIZE_HI	DATA size HI byte
6	1	SIZE_LO	DATA size LO byte
7	1..57	DATA	Payload data (transaction data)

CID – channel ID is a logical channel ID and is generated during the handshake phase described below.

CMD – 0x86 for a handshake, 0x83 for data transfer.

SIZE_HI and SIZE_LO – payload data size.

DATA – payload data. If payload data size < 57 bytes, the packet should be aligned to 64 bytes with random data (all 0x00 or 0xFF are allowed).

Continuation packet format:

Offset	Size (bytes)	Name	Description
0	4	CID	Channel ID
4	1	SEQ	Packet sequence: 0x00...0x7F
5	1..59	DATA	Payload data (transaction data)

CID – channel ID, same as in initialization packet.

SEQ - sequence number, starting from 0 and incremented by 1 in each packet.

DATA – payload data.

Maximum payload size for one transaction is 7609 bytes: one initialization packet with 57 bytes of data + 128 continuation packets with 59 bytes each.

4.2 Handshake

Before the data transfer, the client should initiate the handshake phase with server to generate CID for further data transfers. Handshake request packet is the first packet that should be sent by the client to create logical channel.

Handshake request packet is an initialization packet with specific constant CID, CMD, SIZE and DATA fields.

Client handshake request packet:

Name	Value	Description
CID	0xFFFFFFFF	Handshake CID
CMD	0x86	Handshake command
SIZE_HI	0x00	DATA size = 8 bytes
SIZE_LO	0x08	
DATA	xx … xx	NONCE - 8 random bytes

Server handshake response packet:

Name	Value	Description
CID	0xFFFFFFFF	Handshake CID
CMD	0x86	Handshake command
SIZE_HI	0x00	DATA size = 0x11 bytes
SIZE_LO	0x11	
DATA	Client Response	SERVER Response Data - 0x11 bytes

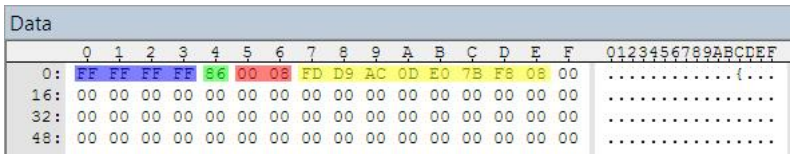
SERVER Response Data description:

Name	Size	Description
NONCE	8 bytes	NONCE from handshake request
CID	4 bytes	CID for further communication
DATA	5 bytes	Could be ignored

Now client and the server have CID (channel ID) value that should be used for all further packets for data transfer.

4.3 Handshake example

Raw data client request captured by USB sniffer:



FF FF FF FF – Handshake CID.

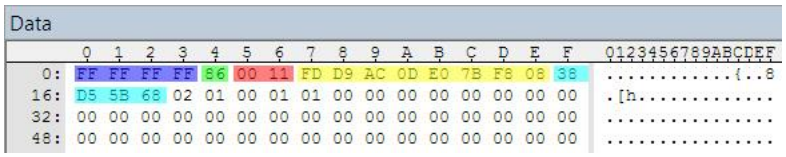
86 – Handshake command.

00 08 – Handshake data size.

FD D9 AC 0D E0 7B F8 08 – handshake NONCE randomly generated by client.

00 … 00 – packet alignment to 64 bytes.

Raw data server response captured by USB sniffer:



```
Data
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
 0:  FF FF FF FF 86 00 11 FD D9 AC 0D E0 7B F8 08 38  ............{..8
16:  D5 5B 68 02 01 00 01 01 00 00 00 00 00 00 00 00  .[h.............
32:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
48:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

FF FF FF FF – Handshake CID.

86 – Handshake command.

00 11 – Handshake response data size.

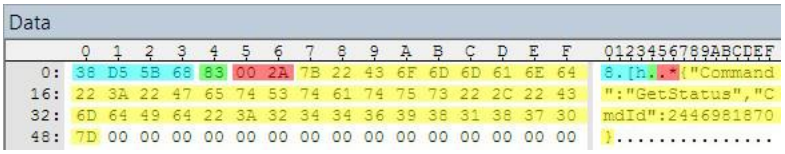
FD D9 AC 0D E0 7B F8 08 – handshake NONCE. Same as in client request.

38 D5 5B 68 – CID generated by server, for further communication.

hidInitChannel(timeout) function from usbhid.js is used for handshake procedure.

4.4 Data transfer example

Server command, raw data captured by USB sniffer:



```
Data
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
 0:  38 D5 5B 68 83 00 2A 7B 22 43 6F 6D 6D 61 6E 64  8.[h..*{"Command
16:  22 3A 22 47 65 74 53 74 61 74 75 73 22 2C 22 43  ":"GetStatus","C
32:  6D 64 49 64 22 3A 32 34 34 36 39 38 31 38 37 30  mdId":2446981870
48:  7D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  }...............
```

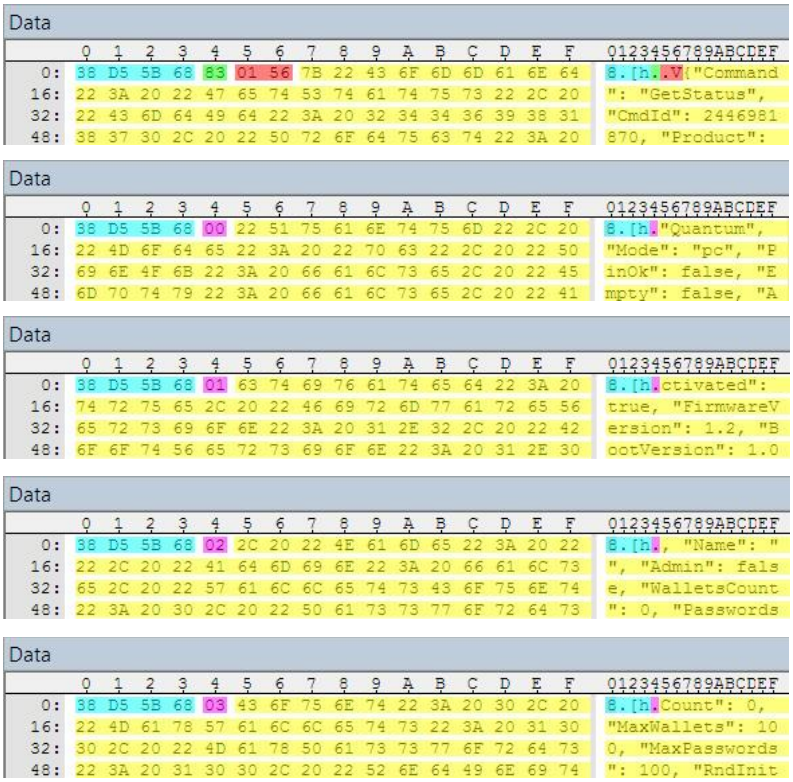
38 D5 5B 68 – Packet CID.

83 – Data transfer command.

00 2A – Payload size.

7B 22 … 30 7D – Payload (Transaction data).

Client response, raw data, initialisation packet:



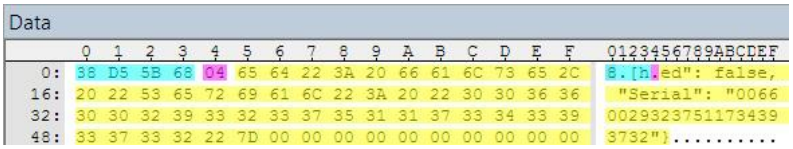
```
Data
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
 0:  38 D5 5B 68 83 01 56 7B 22 43 6F 6D 6D 61 6E 64  8.[h..V{"Command
16:  22 3A 20 22 47 65 74 53 74 61 74 75 73 22 2C 20  ": "GetStatus",
32:  22 43 6D 64 49 64 22 3A 20 32 34 34 36 39 38 31  "CmdId": 2446981
48:  38 37 30 2C 20 22 50 72 6F 64 75 63 74 22 3A 20  870, "Product":

Data
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
 0:  38 D5 5B 68 00 22 51 75 61 6E 74 75 6D 22 2C 20  8.[h."Quantum",
16:  22 4D 6F 64 65 22 3A 20 22 70 63 22 2C 20 22 50  "Mode": "pc", "P
32:  69 6E 4F 6B 22 3A 20 66 61 6C 73 65 2C 20 22 45  inOk": false, "E
48:  6D 70 74 79 22 3A 20 66 61 6C 73 65 2C 20 22 41  mpty": false, "A

Data
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
 0:  38 D5 5B 68 01 63 74 69 76 61 74 65 64 22 3A 20  8.[h.ctivated":
16:  74 72 75 65 2C 20 22 46 69 72 6D 77 61 72 65 56  true, "FirmwareV
32:  65 72 73 69 6F 6E 22 3A 20 31 2E 32 2C 20 22 42  ersion": 1.2, "B
48:  6F 6F 74 56 65 72 73 69 6F 6E 22 3A 20 31 2E 30  ootVersion": 1.0

Data
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
 0:  38 D5 5B 68 02 2C 20 22 4E 61 6D 65 22 3A 20 22  8.[h., "Name": "
16:  22 2C 20 22 41 64 6D 69 6E 22 3A 20 66 61 6C 73  ", "Admin": fals
32:  65 2C 20 22 57 61 6C 6C 65 74 73 43 6F 75 6E 74  e, "WalletsCount
48:  22 3A 20 30 2C 20 22 50 61 73 73 77 6F 72 64 73  ": 0, "Passwords

Data
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
 0:  38 D5 5B 68 03 43 6F 75 6E 74 22 3A 20 30 2C 20  8.[h.Count": 0,
16:  22 4D 61 78 57 61 6C 6C 65 74 73 22 3A 20 31 30  "MaxWallets": 10
32:  30 2C 20 22 4D 61 78 50 61 73 73 77 6F 72 64 73  0, "MaxPasswords
48:  22 3A 20 31 30 30 2C 20 22 52 6E 64 49 6E 69 74  ": 100, "RndInit
```



00 – 04 – Packet sequence.

5 Transaction layer

Transaction layer is the highest layer. Each transaction consists of a command initiated by the client and response sent by the server. Commands and the responses are JSON objects in text format.

Command (request) always consists of two mandatory fields *"Command"* and "*CmdId"*, and some specific parameters.

"CmdId" - unique random value (4 bytes unsigned integer), for each transaction.

Response consists of *"CmdId"* (same value as in request) and *"Command"* field – if command executed OK, or *"Error"* field – if an error occurs. See **Appendix A** for error codes and descriptions.

Only 2 commands could be executed on unauthenticated device (before user enter PIN code):

"GetStatus" – to check PIN status.

"SetTime" – to set current time.

All other commands will return an error.

Some commands, for security reasons, require the user's interaction – device OK button press to confirm command.

5.1 Commands list

Command	PIN	OK Button	
GetStatus	No	No	Get device status
SetTime	No	No	Set real clock time
Restart	Yes	No	Software reset
SetBootMode	Yes	No	Enter boot mode
AddUser	Yes	Yes	Add new user
InitRnd	Yes	Yes	Init random seed
BackupKey	Yes	Yes	Set backup key
BackupBlockRead	Yes	No	Read user memory block
BackupBlockWrite	Yes	No	Write user memory block
GetWallets	Yes	No	Get wallets list
GetPasswords	Yes	No	Get passwords list
GetWalletData	Yes	Yes	Get wallet data
GetWalletPubKey	Yes	No	Get wallet public key (XRP only)
AddWallet	Yes	No	Add new wallet
DelWallet	Yes	Yes	Delete wallet
SignTransaction	Yes	Yes	Sign transaction
GetPasswordData	Yes	Yes	Get password data
Add password	Yes	No	Add new password

DelPassword	Yes	Yes	Delete password
Get2FA	Yes	Yes	Get 2FA value
Add2FA	Yes	No	Add 2FA
Del2FA	Yes	No	Delete 2FA
IncCntr2FA	Yes	Yes	Increment 2FA counter (U2F/HOTP only)
SetCntr2FA	Yes	Yes	Set value for 2FA counter (U2F/HOTP only)
GetSettings	Yes	No	Get device settings
SetSetting	Yes	Yes	Set device settings
ClearMemory	Yes	Yes	Clear user memory

5.2 GetStatus

This command is used to get device status and general information.

Request example:

{"Command": "GetStatus", "CmdId": 1234567890}

Response example:

{"Command": "GetStatus",
"CmdId": 1234567890,
"Product": "Quantum",
"Mode": "pc",
"PinOk": true,
"Empty": false,
"Activated": true,
"FirmwareVersion": 1.2,
"BootVersion": 1.0,
"Name": "User 1",
"Admin": true,
"WalletsCount": 10,
"PasswordsCount": 5,
"MaxWallets": 100,
"MaxPasswords": 100,
"RndInited": false,
"Serial": "00112233445566778899AABB"
}

Where:

"Product" – always **"Quantum"** for QUANTUM device.

"Mode" – always **"pc"** for device command mode.

"PinOk" – **true** if user entered PIN code.

"Empty" – **true** if user memory is empty.

"Activated" – **true** if device is activated.

"FirmwareVersion" – **1.2** current firmware version.

"BootVersion" – **1.0** current bootloader version.

"Name" – **"User 1"**, current user name.

"Admin" – **true** if user have administrator privileges.

"WalletsCount" – **10**, number of wallets for current user.

"PasswordsCount" – **5**, number of passwords for current user.

"MaxWallets" – **100**, max possible wallets for one user.

"MaxPasswords" – **100**, max possible passwords for one user.

"RndInited" – **true** if random seed initialized.

"Serial" – 24 symbols, HEX string unique serial number for current QUANTUM instance.

Possible error codes: none

5.3 SetTime

This command is used to set device real- time clock. The absolute value of the current time is used to calculate 2FA TOTP code. Since QUANTUM does not have a built-in battery this value is not saved after disconnecting the device. It's is required only if 2FA TOTP is going to be used.

Request example:

{"Command": "SetTime", "CmdId": 1234567890,

"Time": 1570882288

}

Where:

"Time" – **1570882288** Unix time (Epoch time, POSIX time), number of seconds that have elapsed since the Unix epoch, that is the time 00:00:00 UTC on 1 January 1970.

Response example:

{"Command": "SetTime", "CmdId": 1234567890}

Possible error codes: 2

5.4 Restart

This command is used to soft reset device. The device is restarted 100ms after the response is sent.

Request example:

{"Command": "Restart", "CmdId": 1234567890}

Response example:

{"Command": "Restart", "CmdId": 1234567890}

Possible error codes: 3

5.5 SetBootMode

Switch device to boot mode for firmware update. The device is restarted in boot mode 100ms after the response is sent.

Request example:

{"Command": "SetBootMode", "CmdId": 1234567890}

Response example:

{"Command": "SetBootMode", "CmdId": 1234567890}

Possible error codes: 3

5.6 AddUser

This command is used to add a new user to the device and enter the PIN code for that user. After the command is sent, the device LCD will start blinking, prompting the user to enter the PIN code using device buttons and LCD.

Request example:

{"Command": "AddUser", "Name": "New User", "Admin": true, "CmdId": 1234567890}

Where:

"Name" – name for new user to be created.

"Admin" – true/false, whether new user will have admin privileges or not. New user will get admin privileges only if it is created from admin user or user memory is empty.

Response example:

{"Command": "AddUser", "CmdId": 1234567890} – command executed ok.

{"Error": "New user not added", "CmdId": 1234567890, "ErrCode": 3} – error executing command.

Possible error codes: 3, 201, 202

5.7 InitRnd

This command is used to initialize internal random seed value using random data from user input (device buttons press). This command should be called once if any of the commands, that

require random numbers are going to be used: *SignTransaction, AddWallet/AddPassword* (with Rnd parameter).

If *SignTransaction* or *AddWallet/AddPassword* (with *Rnd* parameter) command is executed before *InitRnd*, an error is return.

GetStatus command could be used to verify whether random seed is initialized or not, before executing other commands. *RndInited* parameter is **true** if random seed is initialised.

After *InitRnd* command is sent, the device LCD will start blinking, prompting a user to press different buttons on the device.

Request example:

{"Command": "InitRnd", "CmdId": 1234567890}

Response example:

{"Command": "InitRnd", "CmdId": 1234567890}

Possible error codes: 3

5.8 GetWallets

This command is used to get list of wallets.

Request example:

{"Command": "GetWallets", "CmdId": 1234567890}

Response example:

{"Command": "GetWallets", "CmdId": 1234567890, "Max": 100, "Wallets": [

{"Name": "Wallet 1", "Type": "BTC", "Addr": "miYiX7VK8nzE9HyNRF1FfctbaxqQiTKvAE", "Index": 1, "Options": {"Testnet": true}},

{"Name": "Wallet 2", "Type": "LTC", "Addr": mrN2h73EaB9KtPrHZuBTEUy1FdBtpNBkgy", "Index": 2, "Options": {"Testnet": false}},

{"Name": "Wallet 3", "Type": "XRP", "Addr": rNQhsFdBe4rNTeYytXK3FFHkGUYwodaUq4", "Index": 3, "Options": {"Testnet": false}}]

}

Where:

"Max" – maximum number of wallets (same value as in *GetStatus* command).

"Wallets" – an array of objects(wallets).

Each wallet object has:

"Name" – string name.

"Type" – coin type. Possible values: *"BTC", "LTC", "ETH", "DASH", "DOGE", "XRP", "XSN"*

"Addr" – Wallet address.

"Index" – wallets index, to be used in commands (*SignTransaction*, *GetWalletData*, *DelWallet*) to address specific wallet.

"Options" – additional options for each wallet, where: *"Testnet"* – **true** if wallet address is for testnet, and **false** for mainnet.

Possible error codes: 3

5.9 GetPasswords

This command is used to get list of passwords.

Request example:

{"Command": "GetPasswords", "CmdId": 1234567890}

Response example:

```
{"Command": "GetPasswords", "CmdId": 1234567890, "Max": 100, "Passwords": [
    {"Name": "Password 1", "TwoFA": "NONE", "Index": 1},
    {"Name": "Password 2", "TwoFA": "HOTP", "Index": 2},
    {"Name": "Password 3", "TwoFA": "U2F", "Index": 3},
    {"Name": "Password 4", "TwoFA": "TOTP", "Index": 4}]
}
```

Where:

"Max" – maximum number of passwords (same value as in *GetStatus* command).

"Passwords" – an array of objects(passwords).

Each password object has:

"Name" – string name.

"TwoFA" – 2FA type. Possible values: "NONE", "HOTP", "TOTP", "U2F"

"Index" – password index, to be used in commands (*DelPassword, Get2FA, Del2FA, Inc2FA*) to address specific password.

Possible error codes: 3

5.10 GetWalletData

This command is used to get wallet private key and specific data.

After the command is sent, the device LCD will start blinking, prompting the user to press OK or CANCEL buttons on the device to confirm the command.

Request example:

{"Command": "GetWalletData", "Index": 1, "CmdId": 1234567890}

Where:

"Index" – wallet index to get data for.

Response example:

{"Command": "GetWalletData", "CmdId": 1234567890,

"Name": "Wallet 1",

"Type": "BTC",

"Addr": "miYiX7VK8nzE9HyNRF1FfctbaxqQiTKvAE",

"Key": "6B9FC20241EA036B6B33493DE19FCAFAE04900964A9FD8D288EB6EBB068E5F2D",

"Seed": "high wrap letter long parent remember proud hard digital artefact verify volume afraid leopard float fault sugar nephew depart talent race elbow lake pattern",

"WIF": "cRBufFKVRoJgPs9S5ajckraiRWP58BPBTp2dz6UFUKpFXMqVL81n",

"Testnet": true,

"Compressed": true,

"SegWit": false

}

Where:

"Name" – wallet name (same as in *GetWallets* command).

"Type" – coin type (same as in *GetWallets* command).

"Addr" – wallet address (same as in *GetWallets* command).

"Key"/"Seed"/"WIF"/"Secret" – private key in different forms.

"Key", "Seed" and "WIF" - for BTC, LTC, DASH, DOGE, XSN, ETH.

"Key", "Seed" and "Secret" - for XRP.

"Testnet" - **true** if wallet address is for testnet, and **false** for mainnet.

"Compressed" – true if wallet address is generated from compressed public key (for BTC, LTC, DASH, DOGE, XSN, ETH, and could be ignored now).

"SegWit" – true if wallet address is SegWit (for BTC, LTC, DASH, DOGE, XSN and could be ignored now).

"Curve" – for XRP only (always "SECP256K1". Currently only one XRP curve is supported).

"PriveKey" – for XRP only.

If **true** – *"Seed"* and *"Secret"* are empty, and *"Key"* – wallet private key in HEX form.

If **false** – *"Seed"*, *"Key"* and *"Secret"* are XRP account private key.

Possible error codes: 2, 3, 100, 101

5.11 GetWalletPubKey

This command is used to get XRP wallet public key.

XRP public key is required to generate a transaction.

Request example:

{“Command”: “GetWalletPubKey”, “Index”: 3, “CmdId”: 1234567890}

Where:

“Index” – wallet index to get public key for.

Response example:

{“Command”: “GetWalletPubKey”, “CmdId”: 1234567890,

“Name”: “Wallet 1”,

“Type”: “XRP”,

“PubKey”: “6B9FC20241EA036B6B33493DE19FCAFAE04900964A9FD8D288EB6E

BB068E5F2D”

}

Where:

“Name” – wallet name (same as in *GetWallets* command).

“Type” – coin type (same as in *GetWallets* command).

”PubKey” – wallet public key in HEX form.

Possible error codes: 2, 3, 400

5.12 AddWallet

This command is used to add/generate a new wallet.

After command is executed, use *GetWallets* commands to refresh wallets list.

Request example (private key is generated inside of the device):

{“Command”: “AddWallet”,

“Name”: “Wallet 1”,

“Type”: “BTC”,

“Key”: “”,

“Rnd”: true,

“CmdId”: 1234567890

}

Request example (add new wallet with specified private key):

{“Command”: “AddWallet”,

“Name”: “Wallet 1”,

“Type”: “BTC”,

“Key”: “995EEA5F7065223A2022585250222C202241646472223A2022724E5168734664”,

“CmdId”: 1234567890

}

Where:

"Name" – wallet name.

"Type" – coin type (*"BTC", "LTC", "ETH", "DASH", "DOGE", "XRP", "XSN"*).

"Key" – wallet private key. could be any type HEX/Seed/WIF/Secret, device will autodetect key type. Could be empty if "Rnd" is true.

"Rnd" – if **true** – generate random private key inside for new wallet.

If *"Key"* is specified, "Rnd" could be omitted.

Response example:

{"Command": "AddWallet", "CmdId": 1234567890}

Possible error codes: 2, 3, 5, 200, 400, 401, 403, 404, 405, 406

5.13 DelWallet

This command is used to delete the wallet.

After command is sent, the device LCD will start blinking, prompting the user to press OK or CANCEL buttons on the device to confirm command.

After the command is executed, use *GetWallets* commands to refresh wallets list.

Request example:

{"Command": "DelWallet", "Index": 3, "CmdId": 1234567890}

Where:

"Index" – wallet index to be deleted.

Response example:

{"Command": "DelWallet", "CmdId": 1234567890}

Possible error codes: 2, 3, 100, 101, 402

5.14 SignTransaction

This command is used to sign the cryptocurrency transaction.

After the command is sent, the device LCD will start blinking, prompting the user to press OK or CANCEL buttons on the device to confirm the command. Transaction information (amount to send, destination address) is displayed on LCD.

For bitcoin-like coins (BTC, LTC, DASH, DOGE, XSN) each input (UTXO) should be sign by the separate command (request/response). The device will prompt for user interaction only once, when the first input is sent, but will keep a track of the sequence of inputs inside.

Request example:

{“Command”: “SignTransaction”,

“Tx”: “0100000004BC4286F3…88AC0000000001000000”,

“Input”: “1/4”,

“Curve”: “SECP256K1”,

“Index”: 3,

“CmdId”: 1234567890}

Where:

“Index” – wallet index.

“Tx” – HEX string, properly formed transaction to be signed.

“Input” – “input number/total number of inputs”. For example, “1/4” – first input of 4 inputs. ETH and XRP always have 1 input – “1/1”.

“Curve” – cryptographic curve type (for XRP). Currently only *“SECP256K1”* is supported.

Response example:

{“Command”: “SignTransaction”,

“Signature”: “6B4830450221…88D6CC12”,

“CmdId”: 1234567890}

Where:

“Signature” – HEX string, transaction signature for given input (UTXO).

Possible error codes: 2, 3, 100, 101, 407, 408

5.15 GetPasswordData

This command is used to get password specific data.

After the command is sent, device LCD will start blinking, prompting the user to press OK or CANCEL buttons on the device to confirm the command.

Request example:

{“Command”: “GetPasswordData”, “Index”: 1, “CmdId”: 1234567890}

Where:

“Index” – password index to get data for.

Response example:

{“Command”: “GetPasswordData”, “CmdId”: 1234567890,

“Name”: “Password 1”,

“Password”: “user\tpassword\t\r”,

“TwoFA”: “NONE”

}

Where:

"Name" – password name (same as in *GetPasswords* command).

"Password" – password data.

"TwoFA" – 2FA type (*"NONE"*, *"U2F"*, *"HOTP"*, *"TOTP"*).

Possible error codes: 2, 3, 100, 101

5.16 AddPassword

This command is used to add/generate a new password.

After command is executed, use *GetPasswords* commands to refresh passwords list.

Request example (random password is generated inside):

{"Command": "AddPassword",

"Name": "Password 1",

"Rnd": "10",

"Symbols": 7,

"CmdId": 1234567890

}

Request example (add new specified password):

{"Command": "AddPassword",

"Name": "Password 1",

"Password": "user password or any random text",

"CmdId": 1234567890

}

Where:

"Name" – password name.

"Rnd" – random password length. If this field is present – new random password will be generated inside.

"Symbols" – Decimal value, where every bit represents symbols set to be used in password:

1st bit - "abcdefghijklmnopqrstuvwxyz".

2nd bit - "ABCDEFGHIJKLMNOPQRSTUVWXYZ".

3rd bit - "0123456789".

4th bit – special symbols "~!@#$%^&*()-+_=[]{}".

For example, *7* (bits 1, 2 and 3 are set) – password will be generated using digits, small and capital letters, without special symbols.

"Password" – Password data.

Response example:

{“Command”: “AddPassword”, “CmdId”: 1234567890}

Possible error codes: 2, 3, 5, 200, 500, 501, 503, 504, 505, 506, 507

5.17 DelPassword

This command is used to delete the password.

After the command is sent, the device LCD will start blinking, prompting the user to press OK or CANCEL buttons on the device to confirm the command.

After the command is executed, use GetPasswords commands to refresh the passwords list.

Request example:

{“Command”: “DelPassword”, “Index”: 3, “CmdId”: 1234567890}

Where:

“Index” – password index to be deleted.

Response example:

{“Command”: “DelPassword”, “CmdId”: 1234567890}

Possible error codes: 2, 3, 100, 101, 502

5.18 Get2FA

This command is used to get the 2FA code.

After the command is sent, device LCD will start blinking, prompting the user to press OK or CANCEL buttons on the device to confirm the command.

Request example:

{“Command”: “Get2FA”, “Index”: 1,

“Time”: 1572536428,

“CmdId”: 1234567890

}

Where:

“Index” – password index to get 2FA for.

“Time” – current Unix time, same format as in *SetTime* command (for TOTP only).

Response example:

{“Command”: “Get2FA”, “CmdId”: 1234567890,

“Code”: “056109”,

“ValidTime”: 28,

“Counter”: 1

}

Where:

“Code” – 2FA code.

“ValidTime” – 2FA code valid period (in seconds: 0 - 29). After this period 2FA code should be recalculated (for TOTP only).

“Counter” – HOTP counter that was used to calculate 2FA code.

Possible error codes: 2, 3, 100, 101, 508, 509

5.19 Add2FA

This command is used to add 2FA to password.

Request example (for HOTP or TOTP type):

{“Command”: “Add2FA”,

“Index”: 1,

“Type”: “TOTP”,

“Key”: “aaabbbcccddd”

}

Where:

“Index” – password index to add 2FA for.

“Type” – 2FA type (“HOTP”, “TOTP”, “U2F”).

“Key” – 2FA key (for HOTP/TOTP only).

Response example:

{“Command”: “Add2FA”, “CmdId”: 1234567890}

Possible error codes: 2, 3, 5, 200, 508, 510, 512

5.20 Del2FA

This command is used to delete password 2FA.

After the command is sent, device LCD will start blinking, prompting the user to press OK or CANCEL buttons on the device to confirm the command.

After the command is executed, use GetPasswords commands to refresh the passwords list.

Request example:

{"Command": "Del2FA", "Index": 1, "CmdId": 1234567890}

Where:

"Index" – password index to delete 2FA for.

Response example:

{"Command": "Del2FA", "CmdId": 1234567890}

Possible error codes: 2, 3, 100, 101, 511

5.21 IncCntr2FA

This command is used to increment counter for U2F/HOTP 2FA.

Request example:

{"Command": "IncCntr2FA", "Index": 1, "CmdId": 1234567890}

Where:

"Index" – password index to increment 2FA for.

Response example:

{"Command": "IncCntr2FA", "CmdId": 1234567890
"Code": "983245",
"Counter": 2
}

Where:

"Counter" – incremented counter value.

"Code" – 2FA code.

Possible error codes: 2, 3, 100, 101, 508, 509, 513

5.22 SetCntr2FA

This command is used to set counter value for U2F/HOTP 2FA.

Request example:

{"Command": "SetCntr2FA", "Index": 1, "Counter": 10, "CmdId": 1234567890}

Where:

"Index" – password index to set 2FA counter for.

"Counter" – new counter value.

Response example:

{"Command": "SetCntr2FA", "CmdId": 1234567890

"Code": "983245",

"Counter": 2

}

Where:

"Counter" – New counter value.

"Code" – 2FA code.

Possible error codes: 2, 3, 100, 101, 508, 509, 514

5.23 GetSettings

This command is used to read device settings.

Request example:

{"Command": "GetSettings", "CmdId": 1234567890}

Response example:

{"Command": "GetSettings", "CmdId": 1234567890

"Lang": 0,

"DefaultMode": 1,

"AutoLogout": 0,

"PrintDelay": 0,

"ScreenSaver": 60,

"Rotate": 0

}

Where:

"Lang" – device menu language:

0 – English.

1 – Ukrainian.

"DefaultMode" – device default mode after start:

0 – MENU mode.

1 – PC mode.

"AutoLogout" – device auto-logout timeout (seconds).

"PrintDelay" – additional delay (milliseconds) after every symbol printed over USB for MENU mode.

"ScreenSaver" – LCD screen saver timeout (seconds).

"Rotate" – rotate device UI for left hand use:

0 – no rotation.

1 – Rotate.

Possible error codes: 2, 3

5.24 SetSettings

This command is used to write device settings.

After the command is sent, the device LCD will start blinking, prompting the user to press OK or CANCEL buttons on the device to confirm the command.

All the fields are the same as in the GetSettings command.

Request example:

{"Command": "SetSettings", "CmdId": 1234567890,

"Lang": 0,

"DefaultMode": 1,

"AutoLogout": 0,

"PrintDelay": 0,

"ScreenSaver": 60,

"Rotate": 0

}

Response example:

{"Command": "SetSettings", "CmdId": 1234567890}

Possible error codes: 2, 3, 100, 101, 601, 602, 603, 604, 605, 606

5.25 ClearMemory

This command is used to clear all user's memory.

After the command is sent, the device LCD will start blinking, prompting the user to press OK or CANCEL buttons on the device to confirm the command.Current user needs to have admin privileges to execute this command.

Request example:

{"Command": "ClearMemory", "CmdId": 1234567890}

Response example:

{"Command": "ClearMemory", "CmdId": 1234567890}

Possible error codes: 2, 3, 4, 100, 101, 600

5.26 BackupKey

This command is used to set crypto key for backup/restore user memory. After the command is sent, the device LCD will start blinking, prompting the user to enter the crypto key using device buttons and LCD.

Backup/restore should be done by **BackupBlockRead/BackupBlockWrite** commands, by reading/writing one block in each command, starting from address 0, till user memory size ("Size" parameter, returned by this command).

Request example:

{"Command": "BackupKey", "CmdId": 1234567890}

Response example:

{"Command": "BackupKey", "Size": 131584, "BlockSize": 512, "CmdId": 1234567890}

Where:

"Size" – user memory size.

"BlockSize" – maximum block size.

Possible error codes: 3, 300

5.27 BackupBlockWrite

This command is used to write one block from backup file to device user memory.

Request example:

{"Command": "BackupBlockWrite",

"Addr": 0,

"Size": 512,

"Data": "FA05F3…FD43FE",

"CmdId": 1234567890}

Where:

"Size" – block size.

"Addr" – block address.

"Data" – user memory data, hex string.

Response example:

{“Command”: “BackupBlockWrite”, “CmdId”: 1234567890}

Possible error codes: 2, 3, 4, 300, 301, 302

5.28 BackupBlockRead

This command is used to read block of user memory from device.

Request example:

{“Command”: “BackupBlockRead”, “Addr”: 0, “Size”: 512, “CmdId”: 1234567890}

Where:

“Size” – block size.

“Addr” – block address.

Response example:

{“Command”: “BackupBlockRead”,

“Data”: “FA05F3…FD43FE”,

“CmdId”: 1234567890}

Where:

“Data” – user memory data, hex string.

Possible error codes: 2, 3, 300

Appendix A. Command error codes

1 - Unknown command
2 - Invalid parameters
3 - PIN code required
4 - No admin privileges
5 - Rnd seed not initialized

100 - User denied access
101 - User not confirmed

200 - User PIN not set
201 - Invalid name
202 – Entered PIN mismatch

300 – Backup crypto key not set
301 – Backup crypto key error
302 - Backup file format error

400 - Invalid wallet type
401 - Add wallet error
402 - Delete wallet error
403 - Invalid wallet key
404 - Wallet name already exist
405 - Invalid name
406 - Maximum wallets used
407 - Transaction format error
408 - Transaction not allowed

500 - Invalid password
501 - Add password error
502 - Delete password error
503 - Special symbols should be escaped
504 - Invalid password length
505 - Password name already exist
506 - Invalid name
507 - Maximum passwords used
508 - Invalid 2FA type
509 - Can't get 2FA code
510 - Add 2FA error

511 - Delete 2FA error

512 - Invalid 2FA key length

513 - Increment 2FA counter error

514 - Set 2FA counter error

600 - Memory not clean

601 - Language parameter error

602 - Rotate parameter error

603 - Default mode parameter error

604 - Auto logout parameter error

605 - Screen saver parameter error

606 - Print delay parameter error