KaChing

APDU Transaction Protocol

International Application Identifier

The international application identifier (AID) for the KaChing APDU Transaction Protocol is 'A0000008194B43E282BF'.

Abbreviations and Notation

Terminal: The host of the communication session. All commands are sent by the terminal.
Wallet: The bitcoin wallet responding to the commands sent by the terminal.
APDU: Application protocol data unit as defined in ISO/IEC 7816-4.
CLA: Class byte in APDU.
INS: Instruction byte in APDU.
P1-P2: Parameters bytes in APDU.
N_c: Number of bytes in the APDU command data field.
N_e: Maximum number of bytes expected in the APDU response data field.
SW1-SW2: Status bytes in APDU response trailer.
byte: A byte of 8 bits.
bytes: String of bytes.
int8le: Signed integer encoded as 8 bytes, least significant byte first.
varint: Variable length integer in bitcoin format. 1, 3, 5 or 9 bytes.
tx: Transaction in bitcoin format.
SHA-256: Secure hash algorithm as defined in FIPS PUB 180-3.
'XX': Notation using the hexadecimal digits '0' to '9' and 'A' to 'F', equal to XX to the base 16.

Status Bytes

A terminal implementing this protocol must recognize the following status bytes (SW1-SW2) in responses from the wallet.

'9000': Success
'9004': PIN required, 3 or more tries left
'9A04': PIN required, 2 tries left
'9904': PIN required, 1 try left
'9080': Wallet blocked

Commands

If a command response has any status byte other than '9000', the transaction session must be aborted or restarted.

Commands must be issued in the order listed:

1 Get Transaction Template from Wallet
2 Get Public Meta-Key from Wallet
3 Write Transaction to Wallet
4 Get Signatures from Wallet
5 Get Transaction Hash from Wallet
6 Commit Transaction to Wallet

1 Get Transaction Template from Wallet			Required Command
CLA = ’80’	**INS =** ’10’	**P1 =** ’00’	**P2 =** ’00’
N_c = 8 + 1 + PIN length			
Command Data	**int8le:** Transfer amount **byte:** PIN length (bytes) **bytes:** PIN as UTF-8 string in Unicode Normalization Form C		
N_e = 128 or more			

Response Data	**tx:** Transaction template
Expected Status Bytes (SW1-SW2)	’9000’, ’9004’, ’9A04’, ’9904’, ’9080’

Transfer amount is the amount of satoshis to be transferred from the wallet. A negative amount is used to transfer money to the wallet. If transfer amount is zero, it is still recommended that the wallet adds inputs and outputs to the transaction template.

The content of scriptSig of any inputs included in the transaction template is undefined, but must be syntactically correct.

PIN is optional. If the terminal has not acuired a PIN from the user, the PIN length is set to zero.

The transaction template might not fit in a single response. The terminal must repeat the command with empty command data until the entire transaction has been retrieved.

2 Get Public Meta-Key from Wallet			Optional Command
CLA = ’80’	**INS =** ’20’	**P1 =** ’00’	**P2 =** ’00’
N_c = 0			
Command Data	Empty		
N_e = 33			

Response Data	**33 bytes:** Bitcoin public key in compressed format
Expected Status Bytes (SW1-SW2)	’9000’

The public meta-key is only valid for the current transaction. The wallet must implement support for this command.

The terminal should not issue this command if the public meta-key is not needed for the current transaction session.

3 Write Transaction to Wallet			Required Command
CLA = ’80’	**INS =** ’30’	**P1 =** ’00’	**P2 =** ’00’
N_c = Length of transaction chunk			
Command Data	**tx:** Transaction chunk		
N_e = 0			

Response Data	Empty
Expected Status Bytes (SW1-SW2)	’9000’

Terminal must sign for its own inputs before writing the transaction to the wallet.

The transaction can be written in several chunks split at any byte offset by repeating this command.

4 Get Signatures from Wallet			Optional Command
CLA = ’80’	**INS =** ’40’	**P1 =** ’00’	**P2 =** ’00’
N_c = 0			
Command Data	Empty		
N_e = 128 or more			

Response Data	Sequence of **varint:** input number **varint:** scriptSig length (bytes) **bytes:** scriptSig
Expected Status Bytes (SW1-SW2)	’9000’, ’9004’, ’9A04’, ’9904’

Instructs the wallet to sign for its inputs. The terminal must execute this command repeatedly until all signatures has been received.

The terminal must be prepared for the sequence of signatures to be split into chunks at any byte offset.

The terminal must not issue this command if there are no inputs in the transaction that the wallet has to sign for.

5 Get Transaction Hash from Wallet			Required Command
CLA = ’80’	**INS =** ’50’	**P1 =** ’00’	**P2 =** ’00’
N_c = 0			
Command Data	Empty		
N_e = 32			

Response Data	**32 bytes:** Transaction hash
Expected Status Bytes (SW1-SW2)	’9000’

Instructs the wallet to prepare the current transaction for commit.

Response data is the double SHA-256 hash of the transaction.

6 Commit Transaction to Wallet			Required Command
CLA = ’80’	**INS** = ’60’	**P1** = ’00’	**P2** = ’00’
N_c = 0			
Command Data	Empty		
N_e = 0			

Response Data	Empty
Expected Status Bytes (SW1-SW2)	’9000’

This command must only be issued once. After the terminal has committed the transaction to the wallet, it is responsible for publishing the transaction to the blockchain.

The wallet assumes that the transaction will be included in the blockchain, and should respond as quickly as possible to this command.

The terminal must not publish the transaction before this command has completed successfully.