CredaCash™ Transaction API Reference Manual

Creda Software, Inc.

Introduction

CredaCash™ provides a transaction interface to help wallet applications create and submit transactions. The transaction interface consists of:

- A dynamic link or shared library primarily implementing the algorithms used in CredaCash.
- A transaction server connected to the CredaCash network that provides information on past transactions.

Generally, the wallet application sends a JSON command to the dynamic link library and receives binary results, then submits the binary data to a transaction server, receiving back either JSON results or a short string.

The accompanying CredaCash Wallet Developer's Guide provides step-by-step usage instructions. This manual provides a detailed reference. A python example program called "wallet-sim.py" is also available that simulates a working wallet.

Terminology

For convenience, transaction inputs and outputs are referred to as "billets". The "Payor" refers to the person or wallet sending a payment, and the "Payee" refers to the person or wallet receiving a payment.

Billet Values

CredaCash are traded in units of CC. The value of a billet is represented by 64 bit unsigned integers, and by convention the units are femto CC. One CC is therefore represented by the value 2^50, and the maximum value of a single billet is (2^64)-1, which is just less than 16384 CC.

Prime Field Values

Many of the values used in CredaCash are integers in a prime field. Prime field values are computed by taking the result of all operations modulo the prime P. For example, addition is (A + B) mod P, multiplication is (A * B) mod P, etc. CredaCash uses the 254 bit prime:

0x30644e72e131a029b85045b68181585d2833e84879b9709143e1f593f0000001

Prime field numbers are represented internally as 256-bit integers, with the highest two bits always zero.

Elements of Transaction

A transaction can contain up to 18 input and 16 output billets. A transaction consists of the following elements, all of which are publicly published in the transaction:

- merkle-block (64 bits) – the blockchain level associated with the merkle-root and the system parameters minimum-output-value, maximum-output-value and maximum-input-value that were used in the transaction's zero knowledge proof.
- zkkeyid (1 bytes) – id of the proving key used when the transaction's zero knowledge proof was created.
- zkproof (288 bytes) – a zero knowledge proof of the transaction's validity.
- donation (64 bits) – a signed integer representing the amount given to the witnesses that assemble the blockchain.
- nout (1 byte) – the number of output billets in the transaction. The maximum value of nout is 16.

- nin_with_path (1 byte) – the number of input billets that do not have their commitment value published in the transaction (instead, their Merkle paths are internally input into the zero knowledge proof). This is the usual way to specify an input billet, since keeping the input commitment unpublished makes it impossible to link the input back to the output of a prior transaction. The maximum value of nin_with_path is 16.

- nin_with_commitment (1 byte) – the number of input billets with their commitment values publicly published in the transaction. This is not the usual way to specify an input billet since it sacrifices privacy by allowing this input to be linked back to the output of a prior transaction; it can however be useful in certain circumstances, such as chaining together two transactions before the first has cleared. The maximum value of nin_with_commitment is 18, and the maximum value of nin_with_path plus nin_with_commitment is 18.

- a list of the output billets (72 bytes per output)

- a list of the input billets without published commitments (33 bytes per input)

- a list of the input billets with published commitments (65 bytes per input)

Elements of an Output Billet

A billet is created when it is used as the output of a transaction. An output billet consists of the following elements, all of which are published in a transaction:

- address (254 bit prime field value) – payment address. The transaction servers index payments by address, helping the Payee find payments sent to its wallet.

- encrypted-value (64 bits) – an encrypted version of the billet value calculated by the Payee, stored in the blockchain, and then used by the Payee to determine the amount of the payment.

- commitment (254 bit prime field value) – a cryptographic output used to ensure no one can spend a billet that does not exist, tamper with a billet’s value, or spend it without the billet’s spend-secret.

Elements of an Input Billet

A billet is spent when it is used as a transaction input. An input billet consists of the following element which is published in the transaction:

- serial-number (254 bit prime field value) – a cryptographic output used to ensure a billet can only be spent once.

Additional Payee-Generated Billet Elements

In addition to the elements specified above that are published in a transaction, there are a number of elements generated by the Payee and either kept private or shared only with the Payor:

- spend-secret (256 bits random or pseudorandom) – required to spend a billet, and used to generate a payspec. This value is kept private.

- hashed-spend-secret (254 bit prime field value) – while the spend-secret is required to spend a billet, only the hash of the spend-secret is needed to create a payspec and receive payments. This allows the spend-secret to optionally be kept safe by generating and storing it on a secure computer, and only transferring its hash to an online system.

- hashed-spend-spec (256 bits) – for future use: the 256-bit hash of the signing keys and script required to spend the billet.

- enforce-spend-spec-hash (1 bit) – flag indicating whether spending the billet requires a spend-spec. If this value is zero, the billet may be spent with no spend-spec; if it is 1, the spend-spec included when the billet is spent must match the hashed-spend-spec included when the billet is created. This value should currently be set to zero for all billets, since spend-spec's have not yet been implemented.

- payspec (string) – a short text string generated by the "payspec-encode" function that contains all of the information needed for another person to send the Payee a payment.
- destination (254 bit prime field value) – a destination for payments; it is embedded in a payspec and can be extracted using the "payspec-decode" function. Multiple payments can be sent to a single destination.
- sequence-type (1 character) – the requested method the Payor should use when generating payment-number's.

Additional Payor-Generated Billet Elements

The following elements are generated by the Payor and possibly sent to the Payee:

- payment-number (128 bits) – a payment number, which should be generated by the Payor according to the sequence-type requested by the Payee.
- value (64 bits) – the billet value. An encrypted version of this value is published in the blockchain.

Additional System-Generated Billet Elements

The following elements related to a billet are generated by the system, and need to be looked up by the Payor using the "tx-input-query" in order to spend a billet:

- commitment-iv (128 bits) – the lower 128 bits of the merkle-root associated with the merkle-block published in the transaction that created the billet.

- commitment-number (48 bits) – a unique sequential number assigned by the system to all transaction outputs based on the order in which they appear in the blockchain.

- merkle-path (a list of 48 prime field values) – the hash inputs needed to reconstruct the path from the billet's commitment to the root of the Merkle tree containing all commitments.

- merkle-root (254 bit prime field value) – the Merkle tree root value associated with the merkle-block published in the transaction that spends the billets. There is only one merkle-root per transaction, and the merkle-paths above must all be computed relative to that merkle-root.

Conventions

JSON Input: Valid formatted JSON is accepted. Numeric arguments must be integers in decimal or case-insensitive hex format. Hex number must be prefixed by x, X, 0x or 0X and enclosed in double quotes. Decimal numbers greater than 64 bits must be enclosed in double quotes. Double quotes around numbers of 64 or fewer bits are optional but recommended for consistency. For compatibility with python, an upper case L suffix is allowed on all input numbers that are enclosed in double quotes, decimal or hex, and is ignored. Examples of valid numbers:

255, "255", "255L"
"xff", "0xff", "0xFF", "xffL", "0xffL", "0xFFL"

Depending on the use, input values are 1 bit, 64 bits, 128 bits, 256 bits, or 254 bit values in the prime field. Note: on the test network, the donation can be negative. Negative numbers, in decimal or hex format, are allowed by prefixing the value with a minus symbol (-255, "-255", "-255L", "-0xff", "-0xffL"), or they can be provided in two-complement form extended to the bit width of the input value.

JSON Output: The first character of JSON formatted output is always the open brace {. This can be used to distinguish JSON output from a text string. For consistency, JSON numbers are always in lower case hex format prefixed by 0x and enclosed in double-quotes. Negative numbers are always in two's complement form. An example of a JSON output number: "0xff".

Dynamic Link Library

The dynamic link library contains one primary entry point intended for wallet applications. This function is the starting point for all interaction with the transaction API.

int32_t CCTx_JsonCmd(const char *json, char *buffer, const uint32_t bufsize);

Inputs:

- json – string in JSON format containing a function name and its parameters. Detailed descriptions of the functions are given below.

Outputs:

- buffer – an input and output buffer. If required by the JSON command, binary input is taken from this buffer, and then binary, text or JSON output is placed into this buffer.

- bufsize – the total number of bytes available in the buffer for command output.

Returns:

The return value depends on the JSON command, but in general:
zero indicates success
< 0 indicates failure and a text error string is placed into the buffer
> 0 indicates partial success or continue

For functions that return binary data, the first four bytes of output contain the number of output bytes in little endian format.

Function Reference

The functions are listed below roughly in the order in which they would likely be used by a wallet or payment application.

"master-secret-generate"

Generate a wallet master secret using the PBKDF2 key derivation function with the SHA3-512 hash function.

JSON Input:

```
{ "master-secret-generate" : {
        "milliseconds" : milliseconds to hash user password
}}
```

Output: A text string representing the master secret. See the CredaCash Wallet Developer's Guide for important usage notes.

This function generates the 256-bit "salt" input to the PBKDF2 function, determines the number of hash iterations required to run for "milliseconds" on this computer, and stores these values in the output string. These values are encoded into a text string as follows:

The literal "CCMS" is output
The 256-bit salt is output as 44 base58 characters
The iteration count is output as a variable length base58 string
A 5 character "checksum" is output

The base58 encoding uses the characters A-H, J-N, P-Z, a-k, m-z, 1-9 to represent the values 0-57 respectively. The lower bits of the value are output first by outputting the value mod 58, then dividing the value by 58. This is repeated until the desired number of characters have been output (for fixed length outputs) or until the value is zero (for variable length outputs). The "checksum" is the first five base58 characters of the siphash24 of the preceding string.

"master-secret-descramble"

Descrambles a master secret.

JSON Input:

```
{ "master-secret-descramble" : {
        "scrambled-master-secret" : output string from "master-secret-generate"
        "passphrase" : user passphrase
```

Output: a 256-bit master secret encoded as a hex string. See the CredaCash Wallet Developer's Guide for important usage notes.

"tx-parameters-query"

Retrieve the system parameters from a transaction server.

JSON Input:

```
{ "tx-query-create" : {
        "tx-parameters-query" : {}
{}
```

JSON Output:

```
{ "tx-parameters-query-results" : {
        "timestamp" : server's current Unix timestamp
        "query-work-difficulty" : proof-of-work difficulty required for queries
        "tx-work-difficulty" : proof-of-work difficulty required for transactions
        "merkle-tree-oldest-commitment-number" : currently always zero
        "merkle-tree-next-commitment-number" : next commitment-number
```

"hash-spend-secret"

Used by Payee to hash a spend-secret. Use of this function is optional—the next function, "payspec-encode", accepts either a hashed or an unhashed spend-secret.

JSON Input:

```
{ "hash-spend-secret" : {
        "spend-secret" : 256-bit spend-secret
}}
```

JSON Output:

```
{ "hashed-spend-secret" : <val> }
```

"payspec-encode"

Used by Payee to encode a payspec that can be sent to a Payor, posted on the internet, etc.

JSON Input:

```
{ "payspec-encode" : "payspec": {
        "destination"
        OR
        "spend-secret" OR "hashed-spend-secret"
        optional: "enforce-spend-spec-hash" AND "hashed-spend-spec"

        "sequence-type"
        optional: "requested-amount"
}}
```

Output: A text string representing the payspec. The payspec is encoded as follows:

```
        The literal “CC” is output
        A single character (“0”, “1” or “9”) is output for the sequence-type
        The 254-bit destination is output as 44 base58 characters
        If a requested-amount was included:
                The requested-amount is divided by 2 up to 57 times to remove as many
                lower zero bits as possible.
                The number of times it was divided by 2 is output as a base58 character
                The remainder is output as a variable length base58 string
        Two “0” characters are output
        A 5 character “checksum” is output
```

The base58 encoding is the same as "master-secret-generate" (see above).

"payspec-decode"

Used by Payee and Payor to decode a payspec encoded with “payspec-encode” and extract the destination and other values.

JSON Input:

```
{ "payspec-decode" : <text string returned by "payspec-encode"> }
```

JSON Output:

```
{ “payspec” : {
        "destination"
        "sequence-type"
        if present in payspec: "requested-amount"
}}
```

"compute-address"

Used by Payee to determine the payment address based on destination returned from “payspec-decode” and a payment-number sent by Payor or inferred by Payee.

JSON Input:

```
{ "compute-address" : {
        "destination" : value returned by “payspec-decode”
        "payment-number" : value sent by Payor or inferred by Payee
}
```

JSON Output:

```
{ "address" : use in "tx-address-query" to look up payment
  "value-encode-xor" : use to decrypt value returned by "tx-address-query"
}
```

"work-reset"

Resets the proof-of-work fields in the binary output of a library function. This is the first step in creating a proof-of-work, which is required before submitting the binary data to a transaction server.

JSON Input:

```
{ "work-reset" : {
        "timestamp" : current 64-bit Unix time
}}
```

Binary Input: binary output from a prior library function

Binary Output: binary input with proof-of-work values reset.

"work-add"

Updates one of the 8 nonces in the proof-of-work field. All 8 nonces must be valid before the binary data is submitted to a transaction server.

JSON Input:

```
{ "work-add" : {
        "index" : nonce index to update, from 0 to 7
        "iterations" : maximum number of iterations to run
        "difficulty" : the required proof-of-work difficulty
}}
```

Binary Input: binary output from a prior library function

Binary Output: binary input with proof-of-work nonce updated.

Function return value:

-1	the function failed; the output buffer is overwritten with an error message
-2	a valid proof-of-work cannot be found; "work-reset" can be called and the process restarted
0	the nonce contains a valid proof-of-work

1 a valid proof was not found; "work-add" can be called again and the search will be continued

"tx-address-query"

Check if payment has cleared and get its value.

JSON Input:

```
{ "tx-query-create" : {
  "tx-address-query" : {
        "address" : value returned by “compute-address”
        "commitment-number-start" : initially zero; non-zero to continue search
}}
```

The library function returns a binary output which must be submitted to a transaction server after adding a valid proof-of-work with difficulty "query-work-difficulty". The transaction server returns a JSON or a text error message.

JSON Output:

```
{ "tx-address-query-report" : {
        "address" : echo’s input
        "commitment-number-start" : echo’s input
        "more-results-available" : 0 or 1 to indicate more payments at this address
        "tx-address-query-results" : [
list of 1 or more:
        {
                "encrypted-value"
                “commitment-iv"
                "commitment"
                "commitment-number"
        }
] }}
```

Return error messages:

```
        "Not Found"
        "ERROR:<message>"
```

Note: If the wallet calls this function again to look for new payments at the same address, it should set "commitment-number-start" to the last "commitment-number" returned plus one.

"tx-input-query"

Used by Payor to retrieve the information needed to use one or more billets as transaction inputs.

JSON Input:

```
{ "tx-query-create" : {
        "tx-input-query" : { [
list of up to 16:
        {
                "address" : input to "tx-address-query"
                "commitment-number-start" : starting point for search
        }
] }}
```

This function also takes a "commitment-number-start", like “tx-address-query”, so it can also be used to locate new payments. The difference is that “tx-address-query” can return information about multiple payments at an address, while “tx-input-query” can only return information about one payment. “tx-input-query” should only be used when the wallet also requires a merkle-path for the commitment.

The library function returns a binary output which must be submitted to a transaction server after adding a valid proof-of-work with difficulty "query-work-difficulty". The transaction server returns a JSON or a text error message:

JSON Output:

```
{ "tx-input-query-report" : {
        <contents of “tx-parameters-query-results”>
        "donation-per-transaction" : parameter to compute  minimum suggested donation
        "donation-per-byte" : parameter to compute minimum suggested donation
        "donation-per-output" : parameter to compute suggested donation
        "donation-per-input" : parameter to compute minimum suggested donation
        "tx-input-query-results" : {
                "merkle-root"
                "merkle-block"
                "minimum-output-value" : minimum value for output billets
                "maximum-output-value" : maximum value for output billets
                "maximum-input-value" : maximum value for input billets
                "inputs" : [
list of up to 16:
                {
                        "address"
                        "encrypted-value"
                        “commitment-iv"
                        "commitment"
                        "commitment-number"
                        "merkle-path" :
                        [
                                list of 48 hash values
                        ]
                }
] }}}
```

Return error messages:

```
        “Not Found:<index of first input list item that was not found>”
        "ERROR:<message>"
```

Note: the "tx-input-query-results" portion of output is intended to be used as an input to the “tx-create” function. Before using it in “tx-create”, for each input billet, the Payor must delete:

```
        "address"
        "encrypted-value"
```

and must add:

```
        "payment-number"
        "value"
        "spend-secret"
        if used in payspec: "enforce-spend-spec-hash"
        if used in payspec: "hashed-spend-spec"
```

"tx-path-query"

Used by Payor to update the Merkle path of an input previously retrieved with “tx-input-query”.

Not yet implemented.

"tx-create"

Used by Payor to create a transaction.

JSON Input:

```
{ "tx-create" : {
        "tx-pay" : {
                "merkle-root" : value returned by "tx-input-query"
                "merkle-block" : value returned by "tx-input-query"
                "minimum-output-value" : value returned by "tx-input-query"
                "maximum-output-value" : value returned by "tx-input-query"
                "maximum-input-value" : value returned by "tx-input-query"
                "donation" : value computed by formula (see Developer’s Guide)
                "outputs" : [
list of up to 16:
                {
                        "destination" : value returned by "payspec-decode"
                        "payment-number" : value chosen by Payee
                        "value" : value chosen by Payee
                } ]
                "inputs" : [
list of up to 16:
                {
                        "spend-secret" : value used in "payspec-encode"
                        optional: "enforce-spend-spec-hash" : value in "payspec-encode"
                        "payment-number" : value chosen by Payor when billet created
                        "value" : value chosen by Payor when billet created
                        “commitment-iv" : value chosen by Payor when billet created
                        "commitment" : value chosen by Payor when billet created
                        if commitment should not be published in transaction:
                        "commitment-number" : value returned by "tx-input-query"
                        "merkle-path" : list returned by "tx-input-query"
                } ]
                "no-precheck" : 1 to skip validity precheck
                "no-proof" : 1 to not generate zero knowledge proof
                "no-verify" : 1 to not check validity of zero knowledge proof
                "test-make-bad" : 1 to randomly modify a zero knowledge proof input
                "test-use-larger-zkkey" : 1 to randomly choose a larger proof key
}}}
```

If successful, the library function returns 0. On failure, it returns -1 and places an error message in the output buffer.

The library function prepares the transaction and places it into an internal transaction buffer. It can be retrieved using the "tx-to-json" function (which returns the transaction in JSON format), or the "tx-to-wire" function (which returns the transaction in binary “wire” format). There is only one internal transaction buffer that is shared by all functions, so the functions "tx-create" and “tx-from-wire” should not be called by any thread until the wallet is finished working with the contents of the transaction buffer.

"tx-to-json"

Used by Payor to retrieve the transaction prepared by “tx-create”. This would typically be used to extract the commitments and send them to the Payee to use in “tx-address-query”, and extract the serial-number’s to use in "tx-serial-number-query" .

JSON Input:

```
{ "tx-to-json" : {}}
```

JSON Output:

```
{ "tx-pay" : {
        "merkle-root"
        "merkle-block"
        "minimum-output-value"
        "maximum-output-value"
        "maximum-input-value"
        "zkkeyid"
        "zkproof"
        "donation"
        "outputs" : [
list of:
        {
                "destination"
                "payment-number"
                "address"
                "value"
                "encrypted-value"
                “commitment-iv"
                "commitment"
        } ]
        "inputs" : [
list of:
        {
                "destination"
                "payment-number"
                "address"
                "value"
                "encrypted-value"
                “commitment-iv"
                "commitment"
                "spend-secret"
                "enforce-spend-spec-hash"
                "hashed-spend-spec"
                "serial-number"
                if commitment is not publicly published in transaction:
                "commitment-number"
                "merkle-path"
        } ]
}}}
```

"tx-to-wire"

Used by Payor to retrieve the transaction prepared by “tx-create” in binary “wire” format.

JSON Input:

```
{ "tx-to-wire" : {}}
```

Binary Output: the transaction in binary “wire” format.

The library function returns a binary output which can be submitted to a transaction server after adding a valid proof-of-work with difficulty "tx-work-difficulty". The transaction server returns a text result message:

```
        “OK:<next system commitment-number>
        “ERROR:<message>”
```

"tx-from-wire"

Loads a binary transaction into the internal transaction buffer.

JSON Input:

```
{ "tx-from-wire" : {
        "merkle-root" : : value used in "tx-create" (if not, "tx-verify" will fail)
        "minimum-output-value" : value used in "tx-create" (if not, "tx-verify" will fail)
        "maximum-output-value" : value used in "tx-create" (if not, "tx-verify" will fail)
        "maximum-input-value" : value used in "tx-create" (if not, "tx-verify" will fail)
}}
```

Binary Input: the transaction in binary “wire” format.

"tx-verify"

Verifies the zero knowledge proof for the transaction contained in the internal transaction buffer. Note that this only verifies the zero knowledge proof; it does not verify external data such as checking that the merkle root is valid and none of the serial-number’s have been previously spent.

JSON Input:

```
{ "tx-verify" : {}}
```

"tx-dump"

Copies the internal transaction buffer to the function output in diagnostic “dump” format.

JSON Input:

```
{ "tx-dump" : {}}
```

Text Output: the transaction in diagnostic “dump” format.

"tx-serial-number-query"

Used by Payor to check if payment has cleared.

JSON Input:

```
{ "tx-query-create" : {
  "tx-serial-number-query" : {
        "serial-number" : value returned in “tx-to-json”
}}
```

The library function returns a binary output which must be submitted to a transaction server after adding a valid proof-of-work with difficulty "query-work-difficulty". The transaction server returns a text message:

“Indelible”
“Not Found”
“ERROR:<message>”