Semux Light Core Webassembly Documentation

Release v1.0

it_bear

Jun 10, 2020

Contents:

This project is a **WebAssembly** version of the Semux light core library (written in C++, based on libSodium) for the ability to work with Semux cryptographic functions in JavaScript projects.

The project is hosted on GitHub - https://github.com/uno-labs/semux-light-core-wasm

CHAPTER 1

Introduction

This library is essentially intended to create various web versions of the Semux light *HD Wallet*. They can be used both to replace the standard wallet developed by the creators of the Semux core project, as well as for various specialized applications (dApps) based on the Semux ecosystem.

A *wallet* is a tool for creating asymmetric key pairs and digital signatures for transactions in the Semux network. It should have the following main features:

- Random mnemonic phrase generation;
- Creation or recovery an *HD Account* based on a mnemonic phrase;
- Import a private key;
- Generating a random key pair (*Address*);
- Deriving a sequence of key pairs (*HD Addresses*) for the HD Account;
- Finding derived HD Addresses in HD Account;
- Finding *non-HD Addresses* (imported or genearated);
- Generating a message for a transaction;
- Signing transaction messages.

In fact, a fully functional wallet must be able to perform many other functions. Such as, for example, communication with a network node through its API to obtain information necessary for the transaction, or storing wallet data between user sessions in a browser. The implementation of such advanced features is beyond the scope of this lightweight library, designed to perform basic Semux-specific cryptographic operations in the JavaScript environment.

The specificity of Semux algorithms is that they use cryptography on elliptic curves *Ed25519*, and this is why you can't use standard *Web Crypto API* present in modern browsers. Fairly well-known *libSodium* library is most suitable for implementing the algorithms used in Semux. This project makes heavy use of the libSodium.

You can read more about the HD Wallets at the following links:

- Semux Project - https://www.semux.org
- BIP-0039 - https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

- BIP-0032 - https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki
- BIP32-Ed25519 - https://github.com/orogvany/BIP32-Ed25519-java
- SLIP-0100 - https://github.com/satoshilabs/slips/blob/master/slip-0010.md
- BIP-0044 - https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki

1.1 Installation

To build the project, QMake and EMSCRIPTEN compiler are used. The build process is quite complicated, so the compiled files are laid out in the assets at the release section.

Download an archive `uno_semux_light_core.tar.gz` from assets on release page of this project. Then unpack the archive into your project folder.

For use it in the browser project you have to import `UnoSemuxLightCoreWasm.js` into your HTML page:

```
<script src="UnoSemuxLightCoreWasm.js"></script>
```

CHAPTER 2

Terms and Definitions

WebAssembly The WebAssembly (abbreviated *Wasm*) is a software technology that allows you to use code written in C++ in the JavaScript environment.

Wallet A *wallet* is software that stores a set of key pairs of asymmetric cryptography and allows you to perform transaction signing operations using them.

HD Wallet An *Hierarchical Deterministic* wallet is a *wallet* that allows deriving hierarchical chains of key pairs from the initial master seed in a deterministic way.

HD Account An *HD Account* is a very specific intermediate node in the hierarchy of an HD Wallet (defined by the Semux specification), from which all other key pairs are derived.

Address The term *Address* here means an object of the `Addr()` class, which is essentially a key pair.

HD Address An *HD Address* is one of the *Addresses* in the HD Wallet hierarchy.

non-HD Address It is single *Address* not associated with the HD Wallet. It can be obtained by importing a private key or random generation.

> **Note:** This library can simultaneously work with several non-HD addresses, but only with one HD Account (one hierarchy of HD Addresses).

Mnemonic phrase *Mnemonic phrase* (or mnemonic sentence) - is a group of easy to remember words (space separated) for the determinate generation of the master seed (and, accordingly, HD Account) for HD Wallet.

A mnemonic code or sentence is superior for human interaction compared to the handling of raw binary or hexadecimal representations of a wallet master seed. The sentence could be written on paper or spoken over the telephone.

Semux-address Aka *"Hex address"* or *"Recipient address"*. It's a hexadecimal string that is the *"official address"* of some wallet to which you can, for example, transfer a certain amount of cryptocurrency.

In fact, a *Semux-address* is obtained by taking a double hash (Blake2B-SHA256) from the public part of the *Address*.

Nonce *Nonce* is a sequentially increasing and unique integer for the sender address. Max value is 9,223,372,036,854,775,807. Used to make transactions. If you do not know the next *Nonce* for a transaction, then you can get it by contacting the Semux node API.

In the parameters of the methods of this library, the *Nonce* is passed as a string decimal representation.

CHAPTER 3

Getting started

3.1 Important notes

Some methods are **static** and can be called without creating an object. For example:

```
var result = Module.UnoSemux<SomeClass>.<someStaticMethod>();
```

Other methods are members of objects of certain classes. So at first you have to create an object of certain class and then to call its methods. In fact, in this library almost always objects are created by some factory method, e.g.:

```
var myObject = Module.UnoSemux<SomeClass>.<someFactoryMethod>();
var result = myObject.<someMethod>();
```

All methods return a result object that always has two important fields - `error` and `res`:

- `result.error` - if successful, takes an `undefined` value;
- `result.res` - contains the result.

Warning: The methods never throw exceptions related to the logic of the library. But the system exceptions can be thrown nonetheless!

The arguments to the methods, which are essentially integers, are passed as string values. The reason is that JavaScript cannot work with Big Integers.

3.2 Typical usage

First of all you have to include corresponding JavaScript file into your HTML page:

```
<script src="UnoSemuxLightCoreWasm.js"></script>
```

3.2.1 Mnemonic phrase generation

```
<script>
    function NewMnemonicPhrase() {

        var mnemonic_rs = Module.UnoSemuxAccountHD.sNewMnemonic();

        if (typeof mnemonic_rs.error != "undefined") {
            console.log(mnemonic_rs.error);
            return;
        }

        console.log("New mnemonic phrase '" + mnemonic_rs.res + "'");
    }
</script>
```

3.2.2 Import mnemonic phrase

```
<script>
    function ImportMnemonicPhrase() {

        //Mnemonic
        var mnemonic = prompt("Please enter your phrase ");
        var password = "";  //optional

        console.log("HD mnemonic phrase '" + mnemonic + "', password = '" + password␣
↪+ "'");

        //Import HD
        var account_hd_rs = Module.UnoSemuxAccountHD.sImportFromMnemonic(mnemonic,␣
↪password);

        if (typeof account_hd_rs.error != "undefined") {
            console.log(account_hd_rs.error);
            return;
        }

        var account_hd = account_hd_rs.res;

        //Generate next HD address
        console.log("Add next HD address...");
        var next_hd_addr_rs = account_hd.addrAddNextHD();

        if (typeof next_hd_addr_rs.error != "undefined") {
            console.log(next_hd_addr_rs.error);
            return;
        }

        window.next_hd_addr = next_hd_addr_rs.res;

        //Get address as str hex
        var addr_str_hex_rs = window.next_hd_addr.addrStrHex();
        if (typeof addr_str_hex_rs.error != "undefined") {
            console.log(addr_str_hex_rs.error);
            return;
```

(continues on next page)

(continued from previous page)

```
        }

        var addr_str_hex = addr_str_hex_rs.res;

        console.log("New address: " + "0x" + addr_str_hex);
    }
</script>
```

3.2.3 Transaction signature

```
<script>
    function SignTransaction() {

        console.log("New transaction...");

        var d = new Date();

        var network = document.getElementById("transaction_network_source").value;
        var type = document.getElementById("transaction_type_source").value;
        var to = document.getElementById("transaction_to_source").value;
        var value = document.getElementById("transaction_value_source").value;
        var fee = document.getElementById("transaction_fee_source").value;
        var nonce = document.getElementById("transaction_nonce_source").value;
        var data = document.getElementById("transaction_data_source").value;
        var gas = document.getElementById("transaction_gas_source").value;
        var gas_price = document.getElementById("transaction_gas_price_source").value;

        var network_type = Module.UnoSemuxNetworkType.TESTNET;
        if (network == "MAINNET") network_type = Module.UnoSemuxNetworkType.MAINNET;
        else network_type = Module.UnoSemuxNetworkType.TESTNET;

        var transaction_type = Module.UnoSemuxTransactionType.COINBASE;
        if (type == "TRANSFER") transaction_type = Module.UnoSemuxTransactionType.
↪TRANSFER;
        else if (type == "DELEGATE") transaction_type = Module.
↪UnoSemuxTransactionType.DELEGATE;
        else if (type == "VOTE") transaction_type = Module.UnoSemuxTransactionType.
↪VOTE;
        else if (type == "UNVOTE") transaction_type = Module.UnoSemuxTransactionType.
↪UNVOTE;
        else if (type == "CREATE") transaction_type = Module.UnoSemuxTransactionType.
↪CREATE;
        else if (type == "CALL") transaction_type = Module.UnoSemuxTransactionType.
↪CALL;

        var transaction_rs = new Module.UnoSemuxTransaction.sNew(network_type,
            transaction_type,
            String(to),
            String(value),
            String(fee),
            String(nonce),
            String(d.getTime()),
            String(data),
            String(gas),
            String(gas_price));
```

(continues on next page)

(continued from previous page)

```
        if (typeof transaction_rs.error != "undefined") {
            console.log(transaction_rs.error);
            return;
        }

        var transaction = transaction_rs.res;

        console.log("Sign transaction...");
        var sign_rs = window.next_hd_addr.sign1(transaction);

        if (typeof sign_rs.error != "undefined") {
            console.log(sign_rs.error);
            return;
        }

        var sign = sign_rs.res;

        var sign_tx_hash_rs = sign.txHash();

        if (typeof sign_tx_hash_rs.error != "undefined") {
            console.log(sign_tx_hash_rs.error);
            return;
        }
        console.log("Transaction hash '" + sign_tx_hash_rs.res + "'");

        var sign_encode_rs = sign.encode()

        if (typeof sign_encode_rs.error != "undefined") {
            console.log(sign_encode_rs.error);
            return;
        }

        console.log("Transaction sign hex str '" + sign_encode_rs.res + "'");

        document.getElementById("transaction_hash_source").value = sign_tx_hash_rs.
↪res;
        document.getElementById("sign_source").value = sign_encode_rs.res;
    }
</script>
```

CHAPTER 4

AccountHD class

class AccountHD()

An object of this class is not created using the `new` operator, but is returned by the static function *sImportFromMnemonic()* importing a *Mnemonic phrase*.

4.1 Static methods

sNewMnemonic()

Returns A `string` containing generated *Mnemonic phrase*.

Generates a new mnemonic phrase.

Example:

```
var mnemonic_rs = Module.UnoSemuxAccountHD.sNewMnemonic();

if (typeof mnemonic_rs.error != "undefined") {
    console.log(mnemonic_rs.error);
} else {
    console.log("New mnemonic phrase '" + mnemonic_rs.res + "'");
}
```

sImportFromMnemonic(*mnemonic*, *password*)

Arguments

- **mnemonic** (*string*) – A *Mnemonic phrase*.
- **password** (*string*) – A password (can be empty).

Returns An `object` of *AccountHD()* class.

Checks for control sum and imports a *Mnemonic phrase*.
This is essentially a factory method for instantiating an object of *AccountHD()* class.
Using returned object you can further create a sequence of key pairs (objects of *Addr()* class).
Example:

```
function ImportMnemonicPhrase() {

    var mnemonic = prompt("Please enter your mnemonic phrase: ");
    var password = "";  // optional

    // Import mnemonic phrase (transform it into AccountHD)
    var account_hd_rs = Module.UnoSemuxAccountHD.sImportFromMnemonic(mnemonic,␣
↪password);

    if (typeof account_hd_rs.error != "undefined") {
        // If check of mnemonic phrase fails
        console.log(account_hd_rs.error);
    } else {
        var account_hd = account_hd_rs.res;
    }
```

4.2 Class methods

addrAddNextHD()

Returns An `object` of *Addr()* class.

Derives the next key pair (*HD Address*) from the *HD Account*.
Example:

```
// Generate the next HD address
var next_hd_addr_rs = account_hd.addrAddNextHD();

if (typeof next_hd_addr_rs.error != "undefined") {
    console.log(next_hd_addr_rs.error);
} else {
    var next_hd_addr = next_hd_addr_rs.res;
}
```

addrAdd(*address*)

Arguments

- **address** (`Addr`) – An object of *Addr()* class.

Returns `void`.

Add the *non-HD Address* to the collection of Addresses.
You can create such an object of *Addr()* class by *sImportPrivateKeyStrHex()* or *sGenerateNew()* methods.

addrFindByName(*name*)

Arguments

- **name** (*string*) – The name (alias) of the *Address* to search for.

Returns An `object` of *Addr()* class.

Finds the *Address* by its name (alias).

addrFindByHexStr(*hex*)

Arguments

- **hex** (*string*) – A hex form of the *Address* to search for.

Returns An `object` of *Addr()* class.

Finds the *Address* by its HEX representation.

addrHexStrByName(*name*)

Arguments

- **name** (*string*) – The name (alias) of the *Address*.

Returns A `string` containing the HEX representation of an *Address*.

Returns a HEX representation of the *Address* by its name (alias).

addrDeleteByName(*name*)

Arguments

- **name** (*string*) – The name (alias) of the *Address* to be deleted.

Returns `void`.

Deletes the *Address* having the given name.

addrDeleteByHexStr(*hex*)

Arguments

- **name** (*string*) – A hex form of the *Address* to be deleted.

Returns `void`.

Deletes the *Address* by its HEX representation.

CHAPTER 5

Addr class

class Addr()

This class is designed to work with a specific key pair (not with an *HD wallet*).

5.1 Static methods

sImportPrivateKeyStrHex(*hexPrivate*)

Arguments

- **hexPrivate** (*string*) – A HEX form of an importing private key.

Returns An `object` of *Addr()* class.

Imports a private key.

sGenerateNew()

Returns An `object` of *Addr()* class.

Generates a new key pair.

5.2 Class methods

addrStrHex()

Returns A `string` containing a *Semux-address* (without leading '0x').

Method to get a HEX representation of itself (aka *Semux-address*).
Example:

```
// Get address as str hex
var addr_str_hex_rs = next_hd_addr.addrStrHex();

if (typeof addr_str_hex_rs.error != "undefined") {
    console.log(addr_str_hex_rs.error);
} else {
    var addr_str_hex = addr_str_hex_rs.res;
    console.log("HEX address: " + "0x" + addr_str_hex);
}
```

sign1(*transaction*)

Arguments

- **transaction** – An object of *Transaction()* class.

Returns An `object` of *TransactionSign()* class.

Performs a signature of a *Transaction()* object.
Example:

```
var sign_rs = next_hd_addr.sign1(transaction);

if (typeof sign_rs.error != "undefined") {
    console.log(sign_rs.error);
} else {
    var sign = sign_rs.res;
}
```

nonce()

Returns A `string` containing the current *Nonce* (string representation of SINT64 - max value is 9,223,372,036,854,775,807).

Method to get the current *Nonce*, which was set by *setNonce()* method or was incremented by *incNonce()* method.

setNonce(*nonce*)

Arguments

- **nonce** (*string*) – A string representation of *Nonce* to set.

Returns `void`.

Set the *Nonce* for this *Address*.

incNonce()

Returns A `string` containing the incremented *Nonce*.

Method to increment the current *Nonce*.

CHAPTER 6

NetworkType enum

The following constants are used to indicate the type of network:

`Module.UnoSemuxNetworkType.`**`MAINNET`**

`Module.UnoSemuxNetworkType.`**`TESTNET`**

`Module.UnoSemuxNetworkType.`**`DEVNET`**

These constants are used when creating a *`Transaction()`* object.

CHAPTER 7

Transaction class

class Transaction()
An object of *Transaction()* class is created with factory static method *sNew()*.

7.1 Static methods

sNew(*networkType*, *transactionType*, *addressToHex*, *amount*, *fee*, *nonce*, *timestamp*, *dataHex*, *gas*, *gasPrice*)

Arguments

- **networkType** (*NetworkType*) – A type of network.
- **transactionType** (*TransactionType*) – A type of transaction.
- **addressToHex** (*string*) – *Semux-address* in string hexadecimal form.
- **amount** (*string*) – Amount of payment (integer value *in nanosem*).
- **fee** (*string*) – Amount of fee (integer value *in nanosem*).
- **nonce** (*string*) – A *Nonce* (unique and sequential for the sender).
- **timestamp** (*string*) – A timestamp of the transaction (*in milliseconds*).
- **dataHex** (*string*) – Some arbitrary text data in string hexadecimal form.
- **gas** (*string*) – Amount of *gas*.
- **gasPrice** (*string*) – Gas price (integer value *in nanosem*).

Returns object of *Transaction()* class.

Factory method for creating of *Transaction()* class object.
Example:

```
var d = new Date();
var network_type = Module.UnoSemuxNetworkType.TESTNET;
var transaction_type = Module.UnoSemuxTransactionType.TRANSFER;
var to = "0x82c38263217817de2ef28937c7747716eb1e7228";
var data = "0x756E6F2D6C616273206C6967687420776C6C65742064656D6F"; // uno-labs␣
↪light wallet demo
var value = "100000000"; // nanosem
var fee = "5000000";     // nanosem
var nonce = "533";       // Actually, you have to get it from Node API
var gas = "0";
var gas_price = "0";     // nanosem

var transaction_rs = new Module.UnoSemuxTransaction.sNew(
      network_type,
      transaction_type,
      String(to),
      String(value),
      String(fee),
      String(nonce),
      String(d.getTime()),
      String(data),
      String(gas),
      String(gas_price)
);

if (typeof transaction_rs.error != "undefined") {
    console.log(transaction_rs.error);
} else {
    var transaction = transaction_rs.res;
}
```

7.2 Class methods

encode()

Returns An encoded `string` of *Transaction()* object.

Method to get an encoded representation of itself.

CHAPTER 8

TransactionSign class

class TransactionSign()

An object of this class is not created using the `new` operator, but is returned by the *sign1()* or `sign2()` methods of *Addr()* object.

Actually, the *TransactionSign()* objects are storage for the following data:

- encoded transaction data;
- a transaction hash (Blake2B);
- a sign of hash;
- the public key (with no prefix) of the *key pair* with which the signature was made.

8.1 Class methods

txData()

Returns A `string` containing encoded transaction data.

Method to get encoded transaction data.

txHash()

Returns A `string` containing a hash (Blake2B) of the transaction data.

Method to get a hash of the transaction data.
Example:

```
var sign_tx_hash_rs = sign.txHash();

if (typeof sign_tx_hash_rs.error != "undefined") {
    console.log(sign_tx_hash_rs.error);
} else {
    console.log("Transaction hash '" + sign_tx_hash_rs.res + "'");
}
```

sign()

Returns A `string` containing a sign of the transaction data hash.

Method to get a sign of the transaction data hash.

pubKeyNoPrefix()

Returns A `string` containing the public key.

Method to get the public key (with no prefix) of the *key pair* with which the signature was made.

encode()

Returns A `string` containing encoded *TransactionSign()* object.

Encode all data contained in this object in order to prepare before sending to the Semux network.
Example:

```
var sign_encode_rs = sign.encode()

if (typeof sign_encode_rs.error != "undefined") {
    console.log(sign_encode_rs.error);
} else {
    console.log("Transaction sign hex str '" + sign_encode_rs.res + "'");
}
```

CHAPTER 9

TransactionType enum

The following constants are used to indicate the type of transaction:

`Module.UnoSemuxTransactionType.`**`COINBASE`**

`Module.UnoSemuxTransactionType.`**`TRANSFER`**

`Module.UnoSemuxTransactionType.`**`DELEGATE`**

`Module.UnoSemuxTransactionType.`**`VOTE`**

`Module.UnoSemuxTransactionType.`**`UNVOTE`**

`Module.UnoSemuxTransactionType.`**`CREATE`**

`Module.UnoSemuxTransactionType.`**`CALL`**

These constants are used when creating a *`Transaction()`* object.

Index