
Semux Light Core Webassembly Documentation

Release v1.0

it_bear

Aug 10, 2020

Contents:

1	Introduction	3
1.1	Installation	4
2	Terms and Definitions	5
3	Getting started	7
3.1	Important notes	7
3.2	Typical usage	8
4	Wallet class	11
4.1	Static methods	11
4.2	Class methods	12
5	Addr class	15
5.1	Static methods	15
5.2	Class methods	15
6	NetworkType enum	19
7	Transaction class	21
7.1	Static methods	21
7.2	Class methods	22
7.3	Getters	22
8	TransactionSign class	25
8.1	Class methods	25
9	TransactionType enum	27
	Index	29

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 generated);
- 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 Group 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 Wallet The *wallet* that consists of several HD Groups.

HD Account An *HD Account* is a very specific intermediate node in the hierarchy of an HD Group (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 Group hierarchy.

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

Note: This library can simultaneously work with several non-HD addresses, and also with several HD Groups.

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 certain HD Group in 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.

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 `data`:

- `result.error` - if exists, it contains an exception;
- `result.data` - contains result data.

You can check for errors in the following way:

```
<script>  
  function GetRes(aRs)  
  {  
    if (aRs.error)  
    {  
      throw aRs.error;  
    }  
    if (aRs.data)  
    {  
      return aRs.data;  
    }  
    throw 'Unknown result value: ' + JSON.stringify(aRs);  
  }  
</script>
```

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_phrase = GetRes(Module.UnoSemuxWallet.new_mnemonic_phrase());
    console.log("New mnemonic phrase: '" + mnemonic_phrase + "'");
  }
</script>
```

3.2.2 Import mnemonic phrase

```
<script>
  function ImportMnemonicPhrase()
  {
    // Mnemonic phrase from previous example
    var mnemonic = mnemonic_phrase;
    var password = ""; // optional
    console.log("HD mnemonic phrase '" + mnemonic + "', password = '" + password + "'
    ↪");

    // New wallet
    if (!window.semux_wallet)
    {
      console.log("Creating a new wallet");
      window.semux_wallet = GetRes(Module.UnoSemuxWallet.new_wallet());
    }

    // New HD Group from mnemonic phrase (we can add multiple HD groups, each will
    ↪have unique ID)
    console.log("Creating a new HD group...");
    window.semux_hdGroupId = GetRes(window.semux_wallet.add_hd_group(mnemonic,
    ↪password));
    console.log("New HD group ID = " + window.semux_hdGroupId);

    // New HD Address from HD group
    console.log("Creating a new HD Address from HD Group...");
    var hdAddr = GetRes(window.semux_wallet.generate_next_hd_address(window.semux_
    ↪hdGroupId));
```

(continues on next page)

(continued from previous page)

```

    var addrStrHex = GetRes(hdAddr.address());
    console.log("New address: " + "0x" + addrStrHex);
}
</script>

```

3.2.3 Transaction signature

```

<script>
    function SignTransaction()
    {
        // We will assume that the required data is contained in the corresponding_
        ↪ fields of the web form
        var addressStrHex = document.getElementById("hd_address_source").value;
        console.log("Get HD Address from wallet by Semux Address ...");
        var hdAddr = GetRes(window.semux_wallet.find_address(addressStrHex));

        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;

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

        // Transaction type
        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 = GetRes(Module.UnoSemuxTransaction.new_transaction(
            network_type,
            transaction_type,
            String(to),
            String(value),
            String(fee),
            String(nonce),

```

(continues on next page)

(continued from previous page)

```
        String(d.getTime()),
        String(data),
        String(gas),
        String(gas_price)
    ));

    console.log("Sign transaction...");
    var transaction_sign = GetRes(hdAddr.sign_transaction(transaction));

    var transaction_hash = GetRes(transaction_sign.hash());
    console.log("Transaction hash " + transaction_hash + "");

    var transaction_sign_hex_encoded = GetRes(transaction_sign.encode());
    console.log("Transaction sign hex str " + transaction_sign_hex_encoded + "");
}
</script>
```

class Wallet()

An object of this class is not created using the `new` operator, but is returned by the static factory method `new_wallet()`.

4.1 Static methods

new_mnemonic_phrase()

Returns A string containing generated *Mnemonic phrase*.

Generates a new mnemonic phrase.

Example:

```
var mnemonic_phrase = GetRes(Module.UnoSemuxWallet.new_mnemonic_phrase());
console.log("New mnemonic phrase '" + mnemonic_phrase + "'");
```

new_wallet()

Returns An object of *Wallet()* class.

Factory static method to create a new object of *Wallet* class.

Example:

```
if (!window.semux_wallet)
{
    console.log("New wallet");
}
```

(continues on next page)

(continued from previous page)

```
    window.semux_wallet = GetRes(Module.UnoSemuxWallet.new_wallet());  
}
```

4.2 Class methods

add_hd_group (*mnemonic*, *password*)

Arguments

- **mnemonic** (*String*) – A mnemonic phrase to import from.
- **password** (*String*) – An optional password for mnemonic import.

Returns `sInt64` a new Group ID.

Method for creating a new *HD Group* in the *HD Wallet*.

Further, the returned ID is used for operations with *HD Address* es.

```
var semux_hdGroupId = GetRes(window.semux_wallet.add_hd_group(mnemonic,   
    ↪password));  
console.log("New HD group ID = " + semux_hdGroupId);
```

delete_hd_group (*groupId*)

Arguments

- **groupId** (*String*) – The *HD Group* ID to delete from the *HD Wallet*.

Method for deletion the *HD Group* by given *groupId*.

generate_next_hd_address (*groupId*)

Arguments

- **groupId** (*String*) – The ID of the *HD Group*, that is used to get the next *HD Address*.

Returns An object of *Addr()* class.

Method for deriving the next *HD Address* for the *HD Group* by given *groupId* parameter.

Example:

```
var hdAddr = GetRes(window.semux_wallet.generate_next_hd_address(semux_  
    ↪hdGroupId));
```

generate_random_address ()

Returns An object of *Addr()* class.

Method for generating a random *Address* not associated with any *HD Group*.

delete_address (*hexAddress*)

Arguments

- **hexAddress** (*String*) – The *Semux-address* representation of *Address* to delete from the *Wallet*.

Method for deletion the *Address* from the *Wallet* by given *Semux-address*.

find_address (*hexAddress*)

Arguments

- **hexAddress** (*String*) – The *Semux-address* representation of *Address* to find in the *Wallet*.

Returns An object of *Addr()* class.

Method for finding and getting the object of *Addr()* class in the *Wallet* by given *Semux-address*.

class Addr()

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

5.1 Static methods

The class has no static methods.

5.2 Class methods

address()

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

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

Example:

```
//New HD address from HD group
var hdAddr = GetRes(window.semux_wallet.generate_next_hd_address(semux_
  ↪hdGroupId));
var addrStrHex = GetRes(hdAddr.address());
console.log("New address: " + "0x" + addrStrHex);
```

sign_transaction(transaction)**Arguments**

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

Returns An object of *TransactionSign()* class.

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

Example:

```
var transaction = GetRes(Module.UnoSemuxTransaction.new_transaction(  
    network_type,  
    transaction_type,  
    String(to),  
    String(value),  
    String(fee),  
    String(nonce),  
    String(d.getTime()),  
    String(data),  
    String(gas),  
    String(gas_price)  
));  
  
console.log("Sign transaction...");  
var transaction_sign = GetRes(hdAddr.sign_transaction(transaction));  
  
var transaction_hash = GetRes(transaction_sign.hash());  
console.log("Transaction hash '" + transaction_hash + "'");  
  
var transaction_sign_hex_encoded = GetRes(transaction_sign.encode());  
console.log("Transaction sign hex str '" + transaction_sign_hex_encoded + "'");
```

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 *set_nonce()* method or was incremented by *inc_nonce()* method.

set_nonce(nonce)

Arguments

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

Returns void.

Set the *Nonce* for this *Address*.

inc_nonce()

Returns A string containing the incremented *Nonce*.

Method to increment the current *Nonce*.

private_key()

Returns A `string` HEX representation of the *private key* part of this *Address*.

Method to get the HEX representation of the *private key* part of this *Address*.

name ()

Returns A `string` containing the *name* of this *Address* if any name was set by `set_name()` method.

Method to set recognizable name to this *Address*.

set_name (*name*)

Arguments

- **name** (*string*) – Any recognizable name to assign to this Address.

Returns `void`.

Set any recognizable name for this *Address*.

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.

Transaction class

class Transaction()

An object of *Transaction()* class is created with factory static method *new_transaction()* and contains all necessary transaction parameters.

7.1 Static methods

new_transaction (*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 = "0x756E6F2D6C616273206C696768742077616C6C65742064656D6F"; // "uno-labs_
↳light wallet demo" in hex form
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 = GetRes(Module.UnoSemuxTransaction.new_transaction(
    network_type,
    transaction_type,
    String(to),
    String(value),
    String(fee),
    String(nonce),
    String(d.getTime()),
    String(data),
    String(gas),
    String(gas_price)
));

var transaction_sign = GetRes(hdAddr.sign_transaction(transaction));

var transaction_hash = GetRes(transaction_sign.hash());
console.log("Transaction hash " + transaction_hash + "");

var transaction_sign_hex_encoded = GetRes(transaction_sign.encode());
console.log("Transaction sign hex str " + transaction_sign_hex_encoded + "");
```

7.2 Class methods

encode()

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

Method to get an encoded representation of itself.

7.3 Getters

There are also some “getters” methods in the class:

- network_type()
- transaction_type()
- address_to()
- value()
- fee()

- `nonce()`
- `timestamp()`
- `data()`
- `gas()`
- `gas_price()`

TransactionSign class

class TransactionSign()

An object of this class is not created using the `new` operator, but is returned by the `sign_transaction()` method 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

data()

Returns A string containing encoded transaction data.

Method to get encoded transaction data.

hash()

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

Method to get a hash of the transaction data.

sign()

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

Method to get a sign of the transaction data hash.

public_key()

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 transaction_sign = GetRes(hdAddr.sign_transaction(transaction));  
  
var transaction_hash = GetRes(transaction_sign.hash());  
console.log("Transaction hash " + transaction_hash + "");  
  
var transaction_sign_hex_encoded = GetRes(transaction_sign.encode());  
console.log("Transaction sign hex str " + transaction_sign_hex_encoded + "");
```

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.

A

Addr() (class), 15
Address, 5

H

HD Account, 5
HD Address, 5
HD Group, 5
HD Wallet, 5

M

Mnemonic phrase, 5
Module.UnoSemuxNetworkType.DEVNET (global variable or constant), 19
Module.UnoSemuxNetworkType.MAINNET (global variable or constant), 19
Module.UnoSemuxNetworkType.TESTNET (global variable or constant), 19
Module.UnoSemuxTransactionType.CALL (global variable or constant), 27
Module.UnoSemuxTransactionType.COINBASE (global variable or constant), 27
Module.UnoSemuxTransactionType.CREATE (global variable or constant), 27
Module.UnoSemuxTransactionType.DELEGATE (global variable or constant), 27
Module.UnoSemuxTransactionType.TRANSFER (global variable or constant), 27
Module.UnoSemuxTransactionType.UNVOTE (global variable or constant), 27
Module.UnoSemuxTransactionType.VOTE (global variable or constant), 27

N

non-HD Address, 5
Nonce, 6

S

Semux-address, 5

T

Transaction() (class), 21
TransactionSign() (class), 25

W

Wallet, 5
Wallet() (class), 11
WebAssembly, 5