Lity Documentation

CyberMiles

Mar 24, 2019

Contents

This documentation for Lity is still work-in-progress.

CHAPTER 1

Overview

Lity is an open-source language enhanced from Solidity. With new grammar extension, it is highly optimized for enterprise and easy to use for financial industry developers. The flexible interface and unlimited built-in functions can reduce gas cost and execution time. For example, you can now encrypt large-sized data in the blink of an eye without running out of gas anymore.

Since we still supports all features of Solidity, our documentation will focus on unique features realized by Lity.

CHAPTER 2

Contents

2.1 Getting Started

2.1.1 Download the Prebuilt lityc

See Lity releases for the latest release.

Dependencies

- Ubuntu 16.04: `apt install libz3-dev`

Releases

Version	Platform
v1.2.3	Ubuntu 16.04 Static Link ELF Binary
v1.2.2	Ubuntu 16.04 Static Link ELF Binary
v1.1.1	Ubuntu 16.04 macOS Static Link ELF Binary

Or, if you prefer to compile lity yourself, please refer to *developer's guide*.

2.2 Features Overview

2.2.1 Better Security

ERC Checker

Lity can check if your contract's interface is compatible to specific ERC standards, please refer to *ERC contract checker guide*

Overflow Protection

In order to prevent crypto token leak incident like BeautyChain(BEC), Lity offers automatic overflow free guarantee by Lity compiler or Lity's Ethereum virtual machine(EVM). In either solution, an integer overflow would cause the contract execution fail. So contract programmers do not need to worry about overflow detection anymore.

See *here* for more details.

2.2.2 More Flexibility

Provide more flexibility for your EVM with

- *Ethereum Native Interface*.
- *Ethereum Virtual Machine C Interface*.

Ethereum Native Interface

Conventional Smart contracts are compiled to Ethereum byte codes and then executed on EVM. This works well for simple smart contracts. However, for complex contracts such as RSA encrytion/decryption, it is too slow and costs too much gas so that many applications are not feasible. Therefore, we develop a native interface for EVM, called Ethereum Native Interface(ENI).

Smart contracts can call native libraries to speed up execution and save much gas. So performance bottleneck due to EVM can be resolved. Native library are installed with EVM and updated over the air(OTA). See here for more details.

Note that new ENI operations are added without recompiling EVM, rather than pre-compiled contracts.

2.2.3 Business Rule Engine

Lity Rule Language

Lity incorporates rule language(like Drools and Jess) into smart contract. The rules engine shall match patterns(facts) to rules, and resolve potential conflicts. See Lity rule language for reference. Lity's rule grammar is very similar with Drools.

This feature is compatible with official Ethereum Virtual Machine. Because, Lity directly compiles rule language into official EVM byte code.

Check *the rule engine guide* for how to use Lity's rule engine.

2.3 Rule Engine Guide

2.3.1 Introduction

In industry, people often use business rule engine to maintain their business logics and knowledges. Rule engine can detect sepicific conditions and contradiction and execute predefined rules to fix them. Some examples of rule engines are Jess and Drools.

Despite the wide adoption, it is hard to run the rule engine in a publicly verifiable way (which means to run it on Blockchain). To eliminate the gap between traditional business rule engine and Blockchain technology, we decide to implement part of traditional rule engine features (to make it part of Lity). Rule engine related codes are directly compiled into EVM byte codes, so it can also be executed on Ethereum public chain.

Our rule engine's syntax and semantics are directly borrowed from Drools, so it might be a good idea to look through Drools documentation before you use Lity's rule engine. Specifically, chapter Rule Engines and Production Rule Systems (PRS) introduces the basic concept.

2.3.2 Rule Engine Overview

Facts and Working Memory

Facts are data that shall be matched and modified by the rule engine. In Lity, a fact must be a struct stored in storage.

Working memory is a container that stores facts hides behind a contract. To insert/remove facts to/from working memory, we can use `factInsert` and `factDelete` operators.

Rules

A rule defines that `when` certain conditions occur, `then` certain actions should be executed.

In Lity, rules are written in the contract and the syntax is very similar with Drools's.

A rule statement consists of four parts:

1. Rule Name: a string literal which served as the identifier of rule.
2. Rule attributes: optional hints that describe activation behavior of this rule.
3. Filter Statements (a.k.a. *when block*): one or more conditional statements describe which set of facts should be captured.
4. Action Statements (a.k.a. *then block*): one or more statements to execute on matched objects (which are captured in *when block*.)

A contract with a rule definition looks like this:

```
contract C {
    rule "ruleName"
    // Rule attributes
    when {
        // Filter Statements
    } then {
        // Action Statements
    }
}
```

Rule Attributes

salience

default: 0

type: integer literal

Salience specifies the priority of rules in the Activation queue. Higher salience indicates higher priority. Activation order of rules with the same salience would be arbitrary.

In constrast to Drools, Lity does not support dynamic salience.

Due to Solidity parser issue, current salience value cannot be negative, but this shall be resolved in the future.

no_loop

default: false

type: bool literal

`no_loop` forbids a rule to activate itself with the same set of facts. This is for the purpose of preventing infinite loop.

lock_on_active

default: false

type: bool literal

`lock_on_active` forbids a rule to be activated more than once with the same set of facts. This is stronger than `no_loop` because it also prevent the reactivation of the rule even if it is caused by other rule's then-part.

Filter Statements(when)

When part is composed of patterns (which are explained later). When part specifies conditions which set of facts should be activated. If all pattern conditions are met, then part shall be executed for this set of facts.

Pattern

A pattern describe a fact (struct) with a set of conditions. It start with pattern binding, which specifies fact identifier refered in this rule scope. After binding, pattern type specifies the type (struct name) of the fact. Then, a set of constraints is descibe conditions of this fact. Constraints must be boolean expressions. See the example below or refer rule grammar for details.

```
p: Person(age >= 65, eligible == true);
```

Above pattern describe that a fact `p` with Person type, and its constraints are its `age` must be greater or equal to `65` and its `eligible` must be `true`.

Action Statements(then)

Then part is composed of normal statements. However, there is a special operator, `update` (explained later), which might be useful in this part.

Due to Solidity compiler issue, variable declaration statement is not supported yet in then block. But this shall be resolved in the future.

The update operator

`update object` will inform the rule engine that this object may be modified and rules may need to be reevaluated. In current implementation, all rules and facts are reevaluated even for the objects that was not updated. So conditions should be taken care when `update` is used in any rule.

A simple Example

Let's start with a simple example to explain how a rule works. This example pays Ether to old people.

```
rule "payPension" when {
  p: Person(age >= 65, eligible == true);
  b: Budget(amount >= 10);
} then {
  p.addr.transfer(10);
  p.eligible = false;
  b.amount -= 10;
}
```

Above is a rule which pays money to old people if the budget is still enough. `payPension` is the identifier of the rule. There are two patterns in this rule: `p` and `b`. `Person(age >= 65, eligible == true)` describes a person who is at least 65 years old and is eligible for receiving the pension. The `p:` syntax means to bind the qualified person to identifier `p`, so we can refer the person in then-block. `b:  Budget(amount >= 10)` describes the budget must have enough amount. (`10` in this case)

If the rule engine found a person and a budget satisfies above requirements, the then part will be executed. In then part, we modify eligiblity of the person to prevent this rule being applied for the same person again. In addition, pension is sent to the person from the budget.

For full source code of this example, refer the section Rule Examples.

Rule Inheritance

Rules can also be inherited with attributes. This only applies for when part and only supports single inheritance. Refer to section Rule Examples for details.

2.3.3 Rule Examples

This section illustrates more use cases for Lity rule engine.

Pay Pension

This example has already been described in section Rules. The complete contract is below.

```
contract AgePension {
    struct Person {
        int age;
        bool eligible;
        address addr;
    }

    struct Budget {
        int amount;
    }

    mapping (address => uint256) addr2idx;
    Person[] ps;
    Budget budget;

    constructor () public {
        factInsert budget;
        budget.amount = 100;
    }

    function addPerson(int age) public {
        ps.push(Person(age, true, msg.sender));
        addr2idx[msg.sender] = factInsert ps[ps.length-1];
    }

    function deletePerson() public {
        factDelete addr2idx[msg.sender];
    }

    function pay() public {
        fireAllRules;
    }

    function () public payable { }

    rule "payPension" when {
        p: Person(age >= 65, eligible == true);
        b: Budget(amount >= 10);
    } then {
        p.addr.transfer(10);
        p.eligible = false;
        b.amount -= 10;
    }
}
```

A user must use `factInsert` add himself (an instance of `Person`) in order to make the rule engine aware of this data. (written in function `addPerson`) The operator `factInsert` returns an `uint256`. This is where the fact resides in the storage, and this address is recorded in mapping `addr2idx`. The user will be able to remove himself from the engine by `factDelete` with the fact storage address. (written in function `deletePerson`)

The rule `payPension` decribes that gives everyone more than age 65. (already explained in section Rules)

The age pension is paid when `fireAllRules` is executed. (written in function `pay`) `fireAllRules` triggers the rules to find matched rules and apply then part for the corresponding facts.

Fibonacci numbers

Here we demostrate how to use rule engine to calculate fibonacci numbers.

First, we define a struct to represent a fibonacci number:

```
struct E {
    int256 index;
    int256 value;
}
```

The struct has two members. `index` records the index of this fibonacci number, and `value` records the value of the fibonacci number. If the `value` is unknown, we set it to `-1`.

We can now define a rule representing fibonacci number's recurrence relation: $f_n = f_{n-1} + f_{n-2}$.

```
rule "buildFibonacci" when {
    x1: E(value != -1, i1: index);
    x2: E(value != -1, index == i1+1, i2: index);
    x3: E(value == -1, index == i2+1);
} then {
    x3.value = x1.value+x2.value;
    update x3;
}
```

Note that the `update x3;` inside rule's RHS is essential; the update statement informs the rule engine that the value of `x3` has been updated, and all future rule match should not depend on the old value of it.

Let's insert initial terms and unknown fibonacci numbers into working memory

```
// es is a storage array storing `E`
es.push(E(0, 0));
factInsert es[es.length - 1];
es.push(E(1, 1));
factInsert es[es.length - 1];
for (int i = 2 ; i < 10 ; i++) {
    es.push(E(i, -1));
    factInsert es[es.length - 1];
}
```

Working memory now contains f_0, f_1, ... , f_{10}. And only f_0 and f_1's value are known. We can now use `fireAllRules` statement to start the rule engine, and all fibonacci numbers should be calculated accordingly.

Complete source of the contract:

```
contract C {
    struct E {
        int256 index;
        int256 value;
    }

    rule "buildFibonacci" when {
        x1: E(value != -1);
        x2: E(value != -1, index == x1.index+1);
        x3: E(value == -1, index == x2.index+1);
    } then {
        x3.value = x1.value+x2.value;
        update x3;
    }
```

(continues on next page)

(continued from previous page)

```
    E[] es;

    constructor() public {
        es.push(E(0, 0));
        factInsert es[es.length - 1];
        es.push(E(1, 1));
        factInsert es[es.length - 1];
        for (int i = 2 ; i < 10 ; i++) {
            es.push(E(i, -1));
            factInsert es[es.length - 1];
        }
    }

    function calc() public returns (bool) {
        fireAllRules;
        return true;
    }

    function get(uint256 x) public view returns (int256) {
        return es[x].value;
    }

    function () public payable { }
}
```

Examples of salience

If you want some rules to be processed first than other rules (i.e higher priority), `salience` keyword can be used. The bigger the number specified, the higher the priority it have.

```
rule "test1" salience 20 when {
  p: Person(val >= 10);
} then {
  p.addr.send(1);
  p.val--;
  update p;
}

rule "test2" salience 30 when {
  p: Person(val >= 20);
} then {
  p.addr.send(2);
  p.val--;
  update p;
}
```

In the above example, the second rule will have higher priority.

Examples of no_Loop and lock_on_active

Sometimes you may want to update a fact but the activation of the same rule by the same set of fact is not desired.

```
rule "test" when {
  p: Person(age >= 20);
} then {
  p.age++;
  p.addr.send(1);
  update p;
}
```

If you tried to `fireAllRules`, the above rule may keep firing (until `p.age` overflows). To make it fire only once for each `fireAllRules`, we can use `no_loop` keyword.

```
rule "test" no_loop true when {
  p: Person(age >= 20);
} then {
  p.age++;
  p.addr.send(1);
  update p;
}
```

Example of rule inheritance

Sometimes constraints of a rule is based on constraints of another rule. In this case, this rule can `extends` another rule.

For example, a department store wants to give elder customers 10 percent discount and their cars free parking. The discount rule is described as below.

```
rule "Give 10% discount to customers older than 60"
when {
    $customer : Customer( age > 60 );
} then {
    $customer.discount = 10;
}
```

The free parking rule can `extends` the constraint of elder customers (older then 60). Then this rule can be written as below.

```
rule "Give free parking to customers older than 60"
    extends "Give 10% discount to customers older than 60"
when {
    $car : Car ( ownerID == $customer.id );
} then {
    $car.freeParking = true ;
}
```

The rule above (with `extends`) is equivalent to the rule written without `extends`.

```
rule "Give free parking to customers older than 60"
when {
    $customer : Customer( age > 60 );
    $car : Car ( ownerID == $customer.id );
} then {
    $car.freeParking = true ;
}
```

Insurance rating

Insurance claim

Consider a travel insurance that provides claim for fight delay. See the table below.

Delay hours	Compensation
4 or more	5000
6 or more	5000 or accountable expense no more than 15000

The first rule (4 hours or more) is represented as below.

```
rule "four hour fix amount" when{
    p: Person()
    f: Flight(delay >= 4, id == p.flightID)
} then {
    p.claimAmount = max(5000, p.claimAmount);
}
```

For the second rule (6 hours or more), 5000 dollar compensation is implied in the first rule, so we only need to consider the limited expense here.

```
rule "six hour limited amount" when{
    p: Person()
    f: Flight(delay >= 6, id == p.flightID)
} then {
    p.claimAmount = max(min(p.delayExpense, 15000), p.claimAmount);
}
```

Cashier

In the simplest way, cashier sum up all item prices for the amount. Consider restaurants for example, a hamburger costs 50 dollars and a drink costs 30 dollars, and these sum up to 80 dollars. This summation rule could be simply represented as below.

```
rule "Burger"
salience 10
lock_on_active
when{
    b: Burger();
    bl: Bill();
} then {
    bl.amount += 50;
}

rule "Drink"
salience 10
lock_on_active
when{
    d: Drink();
    bl: Bill();
} then {
    bl.amount += 30;
}
```

However, many restaurants offer meal combo discount. For example, a drink with a hamburger is discounted for 10 dollars. With rule engine, this discount rule can be automatically applied as below.

```
rule "Combo" when{
    b: Burger(combo==-1);
    d: Drink(combo==-1);
    bl: Bill();
} then {
    b.combo = bl.nCombo;
    d.combo = bl.nCombo;
    bl.nCombo++;
    bl.amount -= 10;
    update b;
    update d;
}
```

`nCombo` of the bill is the number of combos, and `combo` of a burger/drink denotes the combo number (`-1` denotes no combo) that burger/drink belongs to. Each burger or drink belongs to at most one combo to prevent duplicated discounts.

Credit card cash back

Many credit cards offer conditional cash back. For this example, cash back rates is determined by the total bill amount (of that month). See the rate table below.

Amount	Cash back rate
<5000	0.5%
5000~9999	1.0%
>9999	1.5%

This cash back rule can be represeted as below.

```
rule "poor cash back rate" when{
    b: Bill(amount < 5000);
} then {
    b.cashBack = b.amount * 5 / 1000;
}

rule "sad cash back rate" when{
    b: Bill(amount>=5000, amount < 10000);
} then {
    b.cashBack = b.amount * 10 / 1000;
}

rule "acceptable cash back rate" when{
    b: Bill(amount>=10000);
} then {
    b.cashBack = b.amount * 15 / 1000;
}
```

Tax caculation

In this example, we illustrate how to caculate tax by rule engine. In most countries, tax rates are divides into brackets. That is, certain income range is taxed for corresponding rates. Often, more income indicates higher tax rates.

Take this region for example (For simplicity, we ignore deductions and exemptions for real tax rules. Thus, actual tax rates would be lower.), the corresponding rate table is below.

Net income	Tax rate
0 ~ 540,000	5%
540,000 ~ 1,210,000	12%
1,210,001 ~ 2,420,000	20%
2,420,001 ~ 4,530,000	30%
4,530,001 ~ ∞	40%

For the first tax bracket, net income from 0 to 540000 is taxed for 5%. This is represented as below.

```
rule "first bracket" when{
    p: Person(salary > 0)
} then {
    p.tax += min(540000, p.salary) * 5 / 100;
}
```

Similarly, net income from 540001 to 1210000 is taxed for 12% in the second tax bracket. Note that income 540000 has already been taxed in the first tax bracket, so the amount taxed here should minus 540000.

```
rule "second bracket" when{
    p: Person(salary > 540000)
} then {
    p.tax += (min(1210000, p.salary) - 540000) * 12 / 100;
}
```

In the same way, rest brackets are represented as below.

```
rule "third bracket" when{
    p: Person(salary > 1210000)
} then {
    p.tax += (min(2420000, p.salary) - 1210000) * 20 / 100;
}

rule "fourth bracket" when{
    p: Person(salary > 2420000)
} then {
    p.tax += (min(4530000, p.salary) - 2420000) * 30 / 100;
}

rule "fifth bracket" when{
    p: Person(salary > 4530000)
} then {
    p.tax += (p.salary - 4530000) * 40 / 100;
}
```

Cats

A cat is walking on a number line. Initially it is so hungry that it can't even move. Fortunately, there are some cat foods scattered on the number line. And each cat food can provide some energy to the cat. Whenever the cat's location equal to cat food's location, the cat will immediately eat all the cat foods on that location and gain energy to move forward.

First, we define our fact types:

```
struct Cat {
    uint256 id;
    uint256 energy;
}
struct CatLocation {
    uint256 id;
    uint256 value;
}
struct Food {
    uint256 location;
    uint256 energy;
    bool eaten;
}
```

Here we model the problem in a way similiar to entity-relationship model. `Cat` and `CatLocation` has an one-to-one relationship. Food represents a cat food on the number line, `location` represents its location, `energy` represents how much energy it can provide to Cat. Each unit of energy provides power for the cat to move one unit forward.

Now we can define 2 rules to solve the problem.

```
rule "catEatFood" salience 10
when {
    c1: Cat();
    cl1: CatLocation(id == c1.id);
    f1: Food(location == cl1.value, !eaten);
} then {
    c1.energy += f1.energy;
    update c1;
    f1.eaten = true;
    update f1;
}
```

In the above rule, we first match `Cat` and `CatLocation` using `id`, then match all not yet eaten food that have the same location. If we successfully found a cat whose location equal to the food's location, we let the cat eat the food and tell rule engine that `c1` and `f1`'s value have been modified, so that no food will be eaten twice, for example.

The second rule:

```
rule "catMoves" salience 0
when {
    c1: Cat(energy > 0);
    cl1: CatLocation(id == c1.id);
} then {
    c1.energy--;
    update c1;
    cl1.value++;
    update cl1;
}
```

This rule states that if the cat have positive energy, it can move one unit forward.

`salience` is set so that the cat eat the food whenever its location overlaps with food's location.

Complete source code of the contract:

```
contract C {
    struct Cat {
        uint256 id;
```

(continues on next page)

(continued from previous page)

```
        uint256 energy;
    }
    struct CatLocation {
        uint256 id;
        uint256 value;
    }
    struct Food {
        uint256 location;
        uint256 energy;
        bool eaten;
    }

    // Note that rules appear first have higher priority,
    // so cats won't go through a food without eating it.
    rule "catEatFood" salience 10
    when {
        c1: Cat();
        cl1: CatLocation(id == c1.id);
        f1: Food(location == cl1.value, !eaten);
    } then {
        c1.energy += f1.energy;
        update c1;
        f1.eaten = true;
        update f1;
    }

    rule "catMoves" salience 0
    when {
        c1: Cat(energy > 0);
        cl1: CatLocation(id == c1.id);
    } then {
        c1.energy--;
        update c1;
        cl1.value++;
        update cl1;
    }

    Cat[] cats;
    CatLocation[] catLocations;
    uint256[] factIDs;
    Food[] foods;

    function addCat(uint256 initialLocation) public returns (bool) {
        uint256 newId = cats.length;
        cats.push(Cat(newId, 0));
        catLocations.push(CatLocation(newId, initialLocation));
        factIDs.push(factInsert cats[newId]);
        factIDs.push(factInsert catLocations[newId]);
        return true;
    }

    function addFood(uint256 location, uint256 energy) public returns (bool) {
        foods.push(Food(location, energy, false));
        factIDs.push(factInsert foods[foods.length-1]);
        return true;
    }
```

(continues on next page)

(continued from previous page)

```
    function queryCatCoord(uint256 catId) public view returns (uint256) {
        assert(catLocations[catId].id == catId);
        return catLocations[catId].value;
    }

    function run() public returns (bool) {
        fireAllRules;
        return true;
    }

    function reset() public returns (bool) {
        for (uint256 i = 0; i < factIDs.length; i++)
            factDelete factIDs[i];
        delete cats;
        delete catLocations;
        delete factIDs;
        return true;
    }

    function () public payable { }
}
```

2.3.4 Specifications

Rule Engine Operators

We have three operators to handle facts and working memory:

1. factInsert: add current object as a new fact to working memory.
2. factDelete: remove current object from the working memory.
3. fireAllRules: apply all rules on all facts in working memory.

factInsert

This operator takes a struct with storage data location, evaluates to fact handle, which has type `uint256`. Insert the reference to the storage struct into working memory.

For example:

```
contract C {
  struct fact { int x; }
  fact[] facts;
  constructor() public {
     facts.push(fact(0));
     factInsert facts[facts.length-1]; // insert the fact into working memory
  }
}
```

And note that the following statement cannot be compiled:

```
factInsert fact(0);
```

The reason is that `fact(0)` is a reference with memory data location, which is not persistant thus cannot be inserted into working memory.

For more information about data location mechanism, please refer to solidity's documentation

factDelete

This operator takes a fact handle (uint256) and evaluates to void. Removes the reference of the fact from working memory.

fireAllRules

`fireAllRules` is a special statement that launches lity rule engine execution, it works like drools' `ksession.fireAllRules()` API.

Grammar

Grammar of rule definition:

```
Rule = 'rule' StringLiteral RuleAttributes 'when' '{' RuleLHS '}' 'then' '{' RuleRHS
↪'}'
RuleLHS = ( ( Identifier ':' )? FactMatchExpr ';' )*
FactMatchExpr = Identifier '(' ( FieldExpr ( ',' FieldExpr )* )? ')'
FieldExpr = Expression
RuleRHS = ( Statement | 'update' Identifier ';' )*
RuleAttributes = ( 'no_loop true' | 'lock_on_active true' ( 'salience' DecimalNumber␣
↪) )*
```

Note that some nonterminal symbols are defined in solidity's grammar, including `StringLiteral`, `Identifier`, `Expression`, `Statement`, and `DecimalNumber`.

Rete Network Generation

- Each `FieldExpr` involve more than 1 facts creates a beta node. Otherwise, it creates an alpha node.
- Each nodes corresponding to a dynamic memory array (a data structure which supports lity rule engine runtime execution), these dynamic memory array contains matched fact sets of each node.
- All dynamic memory arrays are reevaluated when `fireAllRules` is called.

2.4 Miscellaneous

2.4.1 Overflow Protections

New Type for Secure Transaction

At the begin of 2018, BEC smart contract were hacked because a human error. There is one and only mathematical operation forgot to use SafeMath Library and overflowed. In the Lity, we introduce a new type *safeuint* in the language to help the Smart Contract developer keep the risk of overflow away.

SafeUint Type

Lity provides a new type *safeuint* which extends from *uint256*, *uint*. *safeuint* is designed for calculating balance of token or any other unsigned integer with overflow protection by default. With this feature, developers don't need to care about how and where to use SafeMath library, because *safeuint* will handle all of the conditions automatically.

Because of the security issue, *safeuint* is not allowed implicit conversion to the other types. If you want to convert, tying to use explicit conversion like the example below:

```
safeuint a = 2000; // OK: safeuint = 2000
safeuint b = a * 12345; // OK: safeuint * uint
// uint c = b; // Error: Implicit conversion is Not Allowed
uint c = uint(a); // OK: Explicit conversion uses strict type conversion
```

In addition, *safeuint* will automatically check overflow during runtime on those operations: add, sub, mul. The behavior just like the SafeMath, but the developer could not to add it manually.

Example 1

```
pragma lity ^1.2.3;

contract TestContract {
    function c() public pure returns (safeuint) {
        safeuint a =␣
↪0x7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff;
        safeuint b =␣
↪0x8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff;
        return (a*b);
    }
}
```

Let compile it with *lityc* and run the bytecode on evm. The evm will raise a INVALID and halt the buggy contract.

```
Missing opcode 0xfepc=00000123 gas=9999999917 cost=10 ERROR: invalid opcode 0xfe
Stack:
00000000  7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
00000001  8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
00000002  8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
00000003  7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
00000004  0000000000000000000000000000000000000000000000000000000000000000

#### LOGS ####
```

We can compile and deploy it:

```
# compile

$ lityc --abi --bin safeuint.sol

======= safeuint.sol:C =======
Binary:
608060405234801561001057600080fd5b5061011e806100206000396000f300608060405260043610603f576000357c01000

Contract JSON ABI
[{"constant":true,"inputs":[],"name":"c","outputs":[{"name":"","type":"uint256"}],
↪"payable":false,"stateMutability":"pure","type":"function"}]
```

(continues on next page)

(continued from previous page)

```
# prepare deploy.js

$ cat deploy.js

bytecode =
↪"0x608060405234801561001057600080fd5b5061011e806100206000396000f30060806040526004361060 3f576000357c
↪"
abi = [{"constant":true,"inputs":[],"name":"c","outputs":[{"name":"","type":"uint256"}
↪],"payable":false,"stateMutability":"pure","type":"function"}]
var testContract = web3.cmt.contract(abi);
var test = testContract.new({
  from: web3.cmt.accounts[0],
  data: bytecode,
  gas: '4700000'
}, function (e, contract){
  if (typeof contract.address !== 'undefined') {
    console.log('Contract mined! address: ' + contract.address + ' transactionHash: '␣
↪+ contract.transactionHash);
  }
})

# deploy

$ travis attach http://localhost:8545

> personal.unlockAccount(cmt.accounts[0], '<YOUR_PASSWORD>')
true
> loadScript('deploy.js')
true
Contract mined! address: 0xab119259ff325f845f8ce59de8ccf63e597a74cd transactionHash:␣
↪0x2c0efff8fa628dfb3f90132d54c1326059d0ad65d938ffd97155282ae94f6d10

# Overflow
> test.c()
0

# The failed transaction consumes all gas
> test.c.sendTransaction({from:cmt.accounts[0], gas:12345678})
"0x8aaef895147ac9d31058eadbff7ee1a65e8adebb626073cd4ed4755b9feb0682"
> cmt.getTransactionReceipt(
↪"0x8aaef895147ac9d31058eadbff7ee1a65e8adebb626073cd4ed4755b9feb0682")
{
  blockHash: "0xff98005eed6c33698954b7de1df12fdae56670eff913e182df223daad9769a99",
  blockNumber: 90,
  contractAddress: null,
  cumulativeGasUsed: 12345678,
  from: "0x7eff122b94897ea5b0e2a9abf47b86337fafebdc",
  gasUsed: 12345678,
  logs: [],
  logsBloom:
↪"0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
  status: "0x0",
  to: "0xb6b29ef90120bec597939e0eda6b8a9164f75deb",
  transactionHash: "0x8aaef895147ac9d31058eadbff7ee1a65e8adebb626073cd4ed4755b9feb0682
↪",
```

(continues on next page)

(continued from previous page)

```
  transactionIndex: 0
}
```

Example 2

Here is another `safeuint` example to multiply two numbers.

```
pragma lity ^1.2.3;

contract TestContract {
    function useSafeuint(safeuint _amountA, safeuint _amountB) public pure returns␣
↪(safeuint) {
        return _amountA * _amountB;
    }
}
```

Then compile and deploy it:

```
$ lityc --abi --bin safeuint.sol

Compile it and deploy it:

======= safeuint.sol:TestContract =======
Binary:
6080604052348015610010576000800fd5b5060f68061001f6000396000f3006080604052600436106003f576000357c010000

Contract JSON ABI
[{"constant":true,"inputs":[{"name":"_amountA","type":"uint256"},{"name":"_amountB",
↪"type":"uint256"}],"name":"useSafeuint","outputs":[{"name":"","type":"uint256"}],
↪"payable":false,"stateMutability":"pure","type":"function"}]

$ travis attach http://localhost:8545

> personal.unlockAccount(cmt.accounts[0], '<YOUR_PASSWORD>')
true

> bytecode =
↪"0x6080604052348015610010576000800fd5b5060f68061001f6000396000f3006080604052600436106003f576000357c01
↪"
> abi = [{"constant":true,"inputs":[{"name":"_amountA","type":"uint256"},{"name":"_
↪amountB","type":"uint256"}],"name":"useSafeuint","outputs":[{"name":"","type":
↪"uint256"}],"payable":false,"stateMutability":"pure","type":"function"}]
> var testContract = web3.cmt.contract(abi);
> var test = testContract.new({
..   from: web3.cmt.accounts[0],
..   data: bytecode,
..   gas: '4700000'
.. }, function (e, contract){
..   if (typeof contract.address !== 'undefined') {
..     console.log('Contract mined! address: ' + contract.address + '␣
↪transactionHash: ' + contract.transactionHash);
..   }
.. })

> test.useSafeuint(2, 2)
```

(continues on next page)

(continued from previous page)

```
4

# The normal transaction with normal gas usage
> tx = test.useSafeuint.sendTransaction(2, 2, {from: cmt.accounts[0], gas:12345678})
"0xd65ddf51d6ad0b83b2fd15816e06d97196bd22afa93371771c497191a82d03ef"
> cmt.getTransactionReceipt(tx)
{
  blockHash: "0xf9736e603d27aedd5abfa1eddd12a3d6676103b2c143c7cc074bc1815ffe221c",
  blockNumber: 310,
  contractAddress: null,
  cumulativeGasUsed: 22010,
  from: "0x7eff122b94897ea5b0e2a9abf47b86337fafebdc",
  gasUsed: 22010,
  logs: [],
  logsBloom:
↪"0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
  status: "0x1",
  to: "0xab119259ff325f845f8ce59de8ccf63e597a74cd",
  transactionHash: "0xd65ddf51d6ad0b83b2fd15816e06d97196bd22afa93371771c497191a82d03ef
↪",
  transactionIndex: 0
}

# Overflow
> test.useSafeuint(2,␣
↪0x8000000000000000000000000000000000000000000000000000000000000000)
0

# The failed transaction consumes all gas
> tx = test.useSafeuint.sendTransaction(2,␣
↪0x8000000000000000000000000000000000000000000000000000000000000000, {from: cmt.
↪accounts[0], gas:12345678})
"0xdc50aac3b91ed6b69df2ec4f891866b9dbe73c361f1e00887c45f9cc2f00caf9"
> cmt.getTransactionReceipt(tx)
{
  blockHash: "0x090c97f1bec942caf803a3b5cbc4723df5f6c0c669f468f69367093de73e1140",
  blockNumber: 315,
  contractAddress: null,
  cumulativeGasUsed: 12345678,
  from: "0x7eff122b94897ea5b0e2a9abf47b86337fafebdc",
  gasUsed: 12345678,
  logs: [],
  logsBloom:
↪"0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
  status: "0x0",
  to: "0xab119259ff325f845f8ce59de8ccf63e597a74cd",
  transactionHash: "0xdc50aac3b91ed6b69df2ec4f891866b9dbe73c361f1e00887c45f9cc2f00caf9
↪",
  transactionIndex: 0
}
```

Overflow Protection

Introduction

In order to prevent crypto token leak incident like BeautyChain(BEC), Lity provides two approaches: improving compiler or Ethereum virtual machine(EVM). The first approach converts every integer operation to SafeMath operations at compile time. The Second approach modifies EVM so that any overflow is detected at runtime.

Lity Compiler

Lity automatically converts integer operations to SafeMath operations, so any overflow would cause a contract execution fail. Compiler options could be set manually to enable/disable this feature.

Lity's Ethereum Virtual machine

While executes a smart contract, Lity's EVM analyzes each integer operation. When an overflow occurs, it will terminate the contract execution and report an error.

Currently, three arithmetic operations are overflow-checked: ADD, SUB and MUL. And overflow checking only applies for 32-byte integers.

Since signed and unsigned overflow must be checked in a different method, each of those integer operations(ADD, SUB and MUL) divides into a signed operation and an unsigned operation. We see original operation(e.g. ADD) as unsigned ones and a new operation(e.g. SADD) as signed one.

Note that this approach only applies for Lity's version of compiler and EVM.

Examples

This example shows how Lity and CyberMiles Virtual Machine helps users to avoid potential security issues of number overflow.

Run a CyberMiles Local Node

Follow the instructions here to have a running CyberMiles local node to deploy a contract.

Compile BEC Contract

There is an overflow issue at this statement `uint256 amount = uint256(cnt) * _value;` of the function `batchTransfer`.

```
pragma lity ^1.2.4;

/**
* @title SafeMath
* @dev Math operations with safety checks that throw on error
*/
library SafeMath {
  function mul(uint256 a, uint256 b) internal constant returns (uint256) {
    uint256 c = a * b;
    assert(a == 0 || c / a == b);
    return c;
```

(continues on next page)

(continued from previous page)

```
  }

  function div(uint256 a, uint256 b) internal constant returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint256 a, uint256 b) internal constant returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal constant returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}

/**
* @title ERC20Basic
* @dev Simpler version of ERC20 interface
* @dev see https://github.com/ethereum/EIPs/issues/179
*/
contract ERC20Basic {
  uint256 public totalSupply;
  function balanceOf(address who) public constant returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

/**
* @title Basic token
* @dev Basic version of StandardToken, with no allowances.
*/
contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) balances;

  /**
  * @dev transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value > 0 && _value <= balances[msg.sender]);

    // SafeMath.sub will throw if there is not enough balance.
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    Transfer(msg.sender, _to, _value);
    return true;
  }
```

(continues on next page)

(continued from previous page)

```
  /**
  * @dev Gets the balance of the specified address.
  * @param _owner The address to query the the balance of.
  * @return An uint256 representing the amount owned by the passed address.
  */
  function balanceOf(address _owner) public constant returns (uint256 balance) {
    return balances[_owner];
  }
}

/**
* @title ERC20 interface
* @dev see https://github.com/ethereum/EIPs/issues/20
*/
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) public constant returns␣
↪(uint256);
  function transferFrom(address from, address to, uint256 value) public returns␣
↪(bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
* @title Standard ERC20 token
*
* @dev Implementation of the basic standard token.
* @dev https://github.com/ethereum/EIPs/issues/20
* @dev Based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/
↪master/smart_contract/FirstBloodToken.sol
*/
contract StandardToken is ERC20, BasicToken {

  mapping (address => mapping (address => uint256)) internal allowed;

  /**
  * @dev Transfer tokens from one address to another
  * @param _from address The address which you want to send tokens from
  * @param _to address The address which you want to transfer to
  * @param _value uint256 the amount of tokens to be transferred
  */
  function transferFrom(address _from, address _to, uint256 _value) public returns␣
↪(bool) {
    require(_to != address(0));
    require(_value > 0 && _value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);

    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    Transfer(_from, _to, _value);
    return true;
  }

  /**
  * @dev Approve the passed address to spend the specified amount of tokens on␣
↪behalf of msg.sender.
```

(continues on next page)

(continued from previous page)

```
  *
  * Beware that changing an allowance with this method brings the risk that someone␣
↪may use both the old
  * and the new allowance by unfortunate transaction ordering. One possible solution␣
↪to mitigate this
  * race condition is to first reduce the spender's allowance to 0 and set the␣
↪desired value afterwards:
  * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
  * @param _spender The address which will spend the funds.
  * @param _value The amount of tokens to be spent.
  */
  function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
  }

  /**
  * @dev Function to check the amount of tokens that an owner allowed to a spender.
  * @param _owner address The address which owns the funds.
  * @param _spender address The address which will spend the funds.
  * @return A uint256 specifying the amount of tokens still available for the␣
↪spender.
  */
  function allowance(address _owner, address _spender) public constant returns␣
↪(uint256 remaining) {
    return allowed[_owner][_spender];
  }
}

/**
* @title Ownable
* @dev The Ownable contract has an owner address, and provides basic authorization␣
↪control
* functions, this simplifies the implementation of "user permissions".
*/
contract Ownable {
  address public owner;
  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

  /**
  * @dev The Ownable constructor sets the original `owner` of the contract to the␣
↪sender
  * account.
  */
  function Ownable() {
    owner = msg.sender;
  }

  /**
  * @dev Throws if called by any account other than the owner.
  */
  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }
```

(continues on next page)

(continued from previous page)

```
  /**
  * @dev Allows the current owner to transfer control of the contract to a newOwner.
  * @param newOwner The address to transfer ownership to.
  */
  function transferOwnership(address newOwner) onlyOwner public {
    require(newOwner != address(0));
    OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }
}

/**
* @title Pausable
* @dev Base contract which allows children to implement an emergency stop mechanism.
*/
contract Pausable is Ownable {
  event Pause();
  event Unpause();

  bool public paused = false;

  /**
  * @dev Modifier to make a function callable only when the contract is not paused.
  */
  modifier whenNotPaused() {
    require(!paused);
    _;
  }

  /**
  * @dev Modifier to make a function callable only when the contract is paused.
  */
  modifier whenPaused() {
    require(paused);
    _;
  }

  /**
  * @dev called by the owner to pause, triggers stopped state
  */
  function pause() onlyOwner whenNotPaused public {
    paused = true;
    Pause();
  }

  /**
  * @dev called by the owner to unpause, returns to normal state
  */
  function unpause() onlyOwner whenPaused public {
    paused = false;
    Unpause();
  }
}

/**
* @title Pausable token
*
```

(continues on next page)

(continued from previous page)

```
* @dev StandardToken modified with pausable transfers.
**/

contract PausableToken is StandardToken, Pausable {
  function transfer(address _to, uint256 _value) public whenNotPaused returns (bool) {
    return super.transfer(_to, _value);
  }
  function transferFrom(address _from, address _to, uint256 _value) public␣
↪whenNotPaused returns (bool) {
    return super.transferFrom(_from, _to, _value);
  }
  function approve(address _spender, uint256 _value) public whenNotPaused returns␣
↪(bool) {
    return super.approve(_spender, _value);
  }
  function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused␣
↪returns (bool) {
    uint cnt = _receivers.length;
    uint256 amount = uint256(cnt) * _value;
    require(cnt > 0 && cnt <= 20);
    require(_value > 0 && balances[msg.sender] >= amount);

    balances[msg.sender] = balances[msg.sender].sub(amount);
    for (uint i = 0; i < cnt; i++) {
      balances[_receivers[i]] = balances[_receivers[i]].add(_value);
      Transfer(msg.sender, _receivers[i], _value);
    }
    return true;
  }
}

/**
* @title Bec Token
*
* @dev Implementation of Bec Token based on the basic standard token.
*/
contract BecToken is PausableToken {
  /**
  * Public variables of the token
  * The following variables are OPTIONAL vanities. One does not have to include them.
  * They allow one to customise the token contract & in no way influences the core␣
↪functionality.
  * Some wallets/interfaces might not even bother to look at this information.
  */
  string public name = "BeautyChain";
  string public symbol = "BEC";
  string public version = '1.0.0';
  uint8 public decimals = 18;

  /**
  * @dev Function to check the amount of tokens that an owner allowed to a spender.
  */
  function BecToken() {
    totalSupply = 7000000000 * (10**(uint256(decimals)));
    balances[msg.sender] = totalSupply;     // Give the creator all initial tokens
  }
```

(continues on next page)

(continued from previous page)

```
  function () {
    //if ether is sent to this address, send it back.
    revert();
  }
}
```

and we could compile it using `lityc`:

Make sure you are in the directory where you downloaded lityc

```
$ mkdir output
$ ./lityc/lityc --abi --bin -o output BEC.sol
$ cat output/BecToken.abi
[{"constant":true,"inputs":[],"name":"name","outputs":[{"name":"","type":"string"}],
↪"payable":false,"stateMutability":"view","type":"function"},{"constant":false,
↪"inputs":[{"name":"_spender","type":"address"},{"name":"_value","type":"uint256"}],
↪"name":"approve","outputs":[{"name":"","type":"bool"}],"payable":false,
↪"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name
↪":"totalSupply","outputs":[{"name":"","type":"uint256"}],"payable":false,
↪"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"name":"_
↪from","type":"address"},{"name":"_to","type":"address"},{"name":"_value","type":
↪"uint256"}],"name":"transferFrom","outputs":[{"name":"","type":"bool"}],"payable
↪":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs
↪":[],"name":"decimals","outputs":[{"name":"","type":"uint8"}],"payable":false,
↪"stateMutability":"view","type":"function"},{"constant":false,"inputs":[],"name":
↪"unpause","outputs":[],"payable":false,"stateMutability":"nonpayable","type":
↪"function"},{"constant":true,"inputs":[],"name":"version","outputs":[{"name":"",
↪"type":"string"}],"payable":false,"stateMutability":"view","type":"function"},{
↪"constant":true,"inputs":[],"name":"paused","outputs":[{"name":"","type":"bool"}],
↪"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs
↪":[{"name":"_owner","type":"address"}],"name":"balanceOf","outputs":[{"name":
↪"balance","type":"uint256"}],"payable":false,"stateMutability":"view","type":
↪"function"},{"constant":false,"inputs":[{"name":"_receivers","type":"address[]"},{
↪"name":"_value","type":"uint256"}],"name":"batchTransfer","outputs":[{"name":"",
↪"type":"bool"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{
↪"constant":false,"inputs":[],"name":"pause","outputs":[],"payable":false,
↪"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name
↪":"owner","outputs":[{"name":"","type":"address"}],"payable":false,"stateMutability
↪":"view","type":"function"},{"constant":true,"inputs":[],"name":"symbol","outputs":[
↪{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":
↪"function"},{"constant":false,"inputs":[{"name":"_to","type":"address"},{"name":"_
↪value","type":"uint256"}],"name":"transfer","outputs":[{"name":"","type":"bool"}],
↪"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,
↪"inputs":[{"name":"_owner","type":"address"},{"name":"_spender","type":"address"}],
↪"name":"allowance","outputs":[{"name":"remaining","type":"uint256"}],"payable
↪":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{
↪"name":"newOwner","type":"address"}],"name":"transferOwnership","outputs":[],
↪"payable":false,"stateMutability":"nonpayable","type":"function"},{"inputs":[],
↪"payable":false,"stateMutability":"nonpayable","type":"constructor"},{"payable
↪":false,"stateMutability":"nonpayable","type":"fallback"},{"anonymous":false,"inputs
↪":[],"name":"Pause","type":"event"},{"anonymous":false,"inputs":[],"name":"Unpause",
↪"type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"name":"previousOwner",
↪"type":"address"},{"indexed":true,"name":"newOwner","type":"address"}],"name":
↪"OwnershipTransferred","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,
↪"name":"owner","type":"address"},{"indexed":true,"name":"spender","type":"address"},
↪{"indexed":false,"name":"value","type":"uint256"}],"name":"Approval","type":"event"}
↪,{"anonymous":false,"inputs":[{"indexed":true,"name":"from","type":"address"},{
↪"indexed":true,"name":"to","type":"address"},{"indexed":false,"name":"value","type":
↪"uint256"}],"name":"Transfer","type":"event"}]
```

(continues on next page)

(continued from previous page)

```
$ cat output/BecToken.bin
```

Deploy contract to Travis locally

After we get contract ABI and bytecode, we could deploy it to Travis chain.

```
# Get Travis console
travis attach http://127.0.0.1:8545

# Deploy contract (in Travis console)
personal.unlockAccount(cmt.accounts[0], '1234');
abi = [{"constant":true,"inputs":[],"name":"name","outputs":[{"name":"","type":"string
↪"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,
↪"inputs":[{"name":"_spender","type":"address"},{"name":"_value","type":"uint256"}],
↪"name":"approve","outputs":[{"name":"","type":"bool"}],"payable":false,
↪"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name
↪":"totalSupply","outputs":[{"name":"","type":"uint256"}],"payable":false,
↪"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"name":"_
↪from","type":"address"},{"name":"_to","type":"address"},{"name":"_value","type":
↪"uint256"}],"name":"transferFrom","outputs":[{"name":"","type":"bool"}],"payable
↪":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs
↪":[],"name":"decimals","outputs":[{"name":"","type":"uint8"}],"payable":false,
↪"stateMutability":"view","type":"function"},{"constant":false,"inputs":[],"name":
↪"unpause","outputs":[],"payable":false,"stateMutability":"nonpayable","type":
↪"function"},{"constant":true,"inputs":[],"name":"version","outputs":[{"name":"",
↪"type":"string"}],"payable":false,"stateMutability":"view","type":"function"},{
↪"constant":true,"inputs":[],"name":"paused","outputs":[{"name":"","type":"bool"}],
↪"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs
↪":[{"name":"_owner","type":"address"}],"name":"balanceOf","outputs":[{"name":
↪"balance","type":"uint256"}],"payable":false,"stateMutability":"view","type":
↪"function"},{"constant":false,"inputs":[{"name":"_receivers","type":"address[]"},{
↪"name":"_value","type":"uint256"}],"name":"batchTransfer","outputs":[{"name":"",
↪"type":"bool"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{
↪"constant":false,"inputs":[],"name":"pause","outputs":[],"payable":false,
↪"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[],"name
↪":"owner","outputs":[{"name":"","type":"address"}],"payable":false,"stateMutability
↪":"view","type":"function"},{"constant":true,"inputs":[],"name":"symbol","outputs":[
↪{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":
↪"function"},{"constant":false,"inputs":[{"name":"_to","type":"address"},{"name":"_
↪value","type":"uint256"}],"name":"transfer","outputs":[{"name":"","type":"bool"}],
↪"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,
↪"inputs":[{"name":"_owner","type":"address"},{"name":"_spender","type":"address"}],
↪"name":"allowance","outputs":[{"name":"remaining","type":"uint256"}],"payable
↪":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{
↪"name":"newOwner","type":"address"}],"name":"transferOwnership","outputs":[],
↪"payable":false,"stateMutability":"nonpayable","type":"function"},{"inputs":[],
↪"payable":false,"stateMutability":"nonpayable","type":"constructor"},{"payable
↪":false,"stateMutability":"nonpayable","type":"fallback"},{"anonymous":false,"inputs
↪":[],"name":"Pause","type":"event"},{"anonymous":false,"inputs":[],"name":"Unpause",
↪"type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"name":"previousOwner",
↪"type":"address"},{"indexed":true,"name":"newOwner","type":"address"}],"name":
↪"OwnershipTransferred","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,
↪"name":"owner","type":"address"},{"indexed":true,"name":"spender","type":"address"},
↪{"indexed":false,"name":"value","type":"uint256"}],"name":"Approval","type":"event"}
↪,{"anonymous":false,"inputs":[{"indexed":true,"name":"from","type":"address"},{
↪"indexed":true,"name":"to","type":"address"},{"indexed":false,"name":"value","type":
↪"uint256"}],"name":"Transfer","type":"event"}]
```

(continues on next page)

(continued from previous page)

```
bytecode =
↪"0x608060405260006003601461010000a81548160ff02191690831515021790555060408051908101604052806 00b815260
↪"
contract = web3.cmt.contract(abi);
c = contract.new(
  {
    from: web3.cmt.accounts[0],
    data: bytecode,
    gas: "4700000"
  },
  function(e, contract) {
    console.log("contract address: " + contract.address);
    console.log("transactionHash: " + contract.transactionHash);
  }
);
```

Test overflow protection

Let's follow this transaction BECTransaction to replay the BEC event.

```
# Call betchTransfer with overflow parameters to test.
> c.batchTransfer.sendTransaction(["0xb4d30cac5124b46c2df0cf3e3e1be05f42119033",
↪"0x0e823ffe018727585eaf5bc769fa80472f76c3d7"],␣
↪0x8000000000000000000000000000000000000000000000000000000000000000, {from:web3.cmt.
↪accounts[0], gas: 1000000})
"0x64c8c5f06bf467917a20b45f52dca8f03613455367c3f36222f27d9fa8992d41"

> t="0x64c8c5f06bf467917a20b45f52dca8f03613455367c3f36222f27d9fa8992d41"
"0x64c8c5f06bf467917a20b45f52dca8f03613455367c3f36222f27d9fa8992d41"

# Our vm consume all of the gas and make this transaction fail, because it will␣
↪overflow and trigger the `throw` opcode.
> cmt.getTransactionReceipt(t)
{
  blockHash: "0xdbae44e6da9087f234a8bac16676e6feee03d02a536cc5b8dc7ccfbceacca3f5",
  blockNumber: 30,
  contractAddress: null,
  cumulativeGasUsed: 1000000,
  from: "0x7eff122b94897ea5b0e2a9abf47b86337fafebdc",
  gasUsed: 1000000,
  logs: [],
  logsBloom:
↪"0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
  root: "0xe34245bd2165cf9be9f4f65ff09d19544fcae32fcd4f301e5f73d1bfa39c78b1",
  to: "0xb6b29ef90120bec597939e0eda6b8a9164f75deb",
  transactionHash: "0x64c8c5f06bf467917a20b45f52dca8f03613455367c3f36222f27d9fa8992d41
↪",
  transactionIndex: 0
}

> cmt.getTransaction(t)
{
  blockHash: "0xdbae44e6da9087f234a8bac16676e6feee03d02a536cc5b8dc7ccfbceacca3f5",
  blockNumber: 30,
```

(continues on next page)

(continued from previous page)

```
  from: "0x7eff122b94897ea5b0e2a9abf47b86337fafebdc",
  gas: 1000000,
  gasPrice: 2000000000,
  hash: "0x64c8c5f06bf467917a20b45f52dca8f03613455367c3f36222f27d9fa8992d41",
  input:
↪"0x83f12fec000000000000000000000000000000000000000000000000000000000000004080000000000016c889a28c1
↪",
  nonce: 1,
  r: "0x123fbcffac70f01bdb2d71b15431e945867c6a6c735194fc4ecb5c462ed2e67a",
  s: "0x656d875d5131e4d91026f4a952bcc4daeeba7ee7bea45c95e7b2949051920a1f",
  to: "0xb6b29ef90120bec597939e0eda6b8a9164f75deb",
  transactionIndex: 0,
  v: "0x49",
  value: 0
}

# If we modify the second parameter from␣
↪0x8000000000000000000000000000000000000000000000000000000000000000 to 0x8, this␣
↪transaction has no overflow issue. Now it can execute normally.
> t=c.batchTransfer.sendTransaction(["0xb4d30cac5124b46c2df0cf3e3e1be05f42119033",
↪"0x0e823ffe018727585eaf5bc769fa80472f76c3d7"], 0x8, {from:web3.cmt.accounts[0],␣
↪gas: 1000000})
"0xef69425c20b1d540be8c8e23bd7565be727b37571c4f47a89a3457b40eb81cbf"

> cmt.getTransaction(t)
{
  blockHash: "0xcb0253add2bb035db135a68a07c4d9a6f7b81e0587df47b86d6456331a04fca4",
  blockNumber: 52,
  from: "0x7eff122b94897ea5b0e2a9abf47b86337fafebdc",
  gas: 1000000,
  gasPrice: 2000000000,
  hash: "0xef69425c20b1d540be8c8e23bd7565be727b37571c4f47a89a3457b40eb81cbf",
  input:
↪"0x83f12fec000000000000000000000000000000000000000000000000000000000000004000000000000000000000000
↪",
  nonce: 2,
  r: "0x71e99ddcc132d738836b117f95aed93bf366d3a8d009742349423a878de0545",
  s: "0x7d4df4f45931c5b48a36567bef4b84db966faf0188bd317379189cf3c61a1005",
  to: "0xb6b29ef90120bec597939e0eda6b8a9164f75deb",
  transactionIndex: 0,
  v: "0x4a",
  value: 0
}

# This transaction only cost 76776 gas.
> cmt.getTransactionReceipt(t)
{
  blockHash: "0xcb0253add2bb035db135a68a07c4d9a6f7b81e0587df47b86d6456331a04fca4",
  blockNumber: 52,
  contractAddress: null,
  cumulativeGasUsed: 76776,
  from: "0x7eff122b94897ea5b0e2a9abf47b86337fafebdc",
  gasUsed: 76776,
  logs: [{
      address: "0xb6b29ef90120bec597939e0eda6b8a9164f75deb",
      blockHash: "0xcb0253add2bb035db135a68a07c4d9a6f7b81e0587df47b86d6456331a04fca4",
      blockNumber: 52,
```

(continues on next page)

(continued from previous page)

```
      data: "0x0000000000000000000000000000000000000000000000000000000000000008",
      logIndex: 0,
      removed: false,
      topics: ["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
↪"0x0000000000000000000000007eff122b94897ea5b0e2a9abf47b86337fafebdc",
↪"0x000000000000000000000000b4d30cac5124b46c2df0cf3e3e1be05f42119033"],
      transactionHash:
↪"0xef69425c20b1d540be8c8e23bd7565be727b37571c4f47a89a3457b40eb81cbf",
      transactionIndex: 0
  }, {
      address: "0xb6b29ef90120bec597939e0eda6b8a9164f75deb",
      blockHash: "0xcb0253add2bb035db135a68a07c4d9a6f7b81e0587df47b86d6456331a04fca4",
      blockNumber: 52,
      data: "0x0000000000000000000000000000000000000000000000000000000000000008",
      logIndex: 1,
      removed: false,
      topics: ["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
↪"0x0000000000000000000000007eff122b94897ea5b0e2a9abf47b86337fafebdc",
↪"0x000000000000000000000000e823ffe018727585eaf5bc769fa80472f76c3d7"],
      transactionHash:
↪"0xef69425c20b1d540be8c8e23bd7565be727b37571c4f47a89a3457b40eb81cbf",
      transactionIndex: 0
  }],
  logsBloom:
↪"0x000000000000000000000000000000000000010000000000000000000000000000000008000000000000000000000000
↪",
  root: "0x0b74d2224a735a6af818664b9ae7cb52ff3575b41dcab616114274bdb311c756",
  to: "0xb6b29ef90120bec597939e0eda6b8a9164f75deb",
  transactionHash: "0xef69425c20b1d540be8c8e23bd7565be727b37571c4f47a89a3457b40eb81cbf
↪",
  transactionIndex: 0
}
```

2.4.2 Everything about Ethereum Native Interface (ENI)

ENI Tutorial

Run Travis Node

What is travis? See here for instructions to start a Travis node.

Make sure to checkout to the *lity* branch.

```
go get github.com/CyberMiles/travis
cd $GOPATH/src/github.com/CyberMiles/travis
git checkout lity
make all
travis node init --home ~/.travis
travis node start --home ~/.travis
```

Get ENI Libraries

See libENI documentation for how to get ENI libraries or build your own ENI libraries.

Install the ENI libraries to Travis environment.

```
mkdir -p ~/.travis/eni/lib/
cp eni_reverse.so ~/.travis/eni/lib/
```

Use ENI in Your Contract

Create a contract file called *Reverse.lity*.

```
pragma lity ^1.2.4;

contract ReverseContract {
  function reverse(string input) public returns(string) {
    string memory output = eni("reverse", input);
    return output;
  }
}
```

Compile Your Contract with Lityc

```
lityc --bin-runtime Reverse.lity
```

```
======= ./Reverse.lity:ReverseContract =======
Binary of the runtime part:
608060405260043610610041576000357c0100000000000000000000000000000000000000000000000000000000900463ff
```

```
lityc --abi Reverse.lity
```

```
======= ./Reverse.lity:ReverseContract =======
Contract JSON ABI
[{"constant":false,"inputs":[{"name":"input","type":"string"}],"name":"reverse",
↪"outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":
↪"nonpayable","type":"function"}]
```

Test Locally

Get our EVM from *here <https://github.com/CyberMiles/go-ethereum>*.

Run with *ABCDE* as input to your contract.

```
evm --code␣
↪608060405260043610610041576000357c0100000000000000000000000000000000000000000000000000000000900463
↪--input␣
↪064767aa00000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000
↪--statdump run
```

```
evm execution time: 19.432062ms
heap objects:       17981
allocations:        2648592
total allocations:  2648592
GC calls:           0
```

(continues on next page)

(continued from previous page)

```
Gas used:            1329

0x000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000
```

According to the ABI of Ethereum contracts, the output could be decode as string *EDCBA*.

```
0000000000000000000000000000000000000000000000000000000000000020    # offset to string
0000000000000000000000000000000000000000000000000000000000000005    # string length
4544434241000000000000000000000000000000000000000000000000000000    # string "EDCBA"
```

Reverse String

This example shows how to use our `eni_revserse.so` to reverse a string on Lity contract.'

Compiler StringReverse Contract

There is a simple contract that leverage ENI to reverse a string.

```
pragma lity ^1.2.4;
contract StringReverse {

      function test() public returns (string) {
              string memory ret;
              ret = eni("reverse", "Hello, world");
              return ret;
      }

      function reverse(string src) public returns (string) {
              string memory ret;
              ret = eni("reverse", src);
              return ret;
      }
}
```

and we could compile it using `lityc`:

```
$ mkdir output
$ lityc --abi --bin -o output StringReverse.sol
$ cat output/StringReverse.abi
[{"constant":true,"inputs":[{"name":"src","type":"string"}],"name":"reverse","outputs
↪":[{"name":"","type":"string"}],"payable":false,"stateMutability":"pure","type":
↪"function"},{"constant":true,"inputs":[],"name":"test","outputs":[{"name":"","type":
↪"string"}],"payable":false,"stateMutability":"pure","type":"function"}]
$ cat output/StringReverse.bin
6080604052348015610010576000080fd5b5061041a806100206000396000f300608060405260043610610004c576000357c010
```

Deploy contract to Travis locally

After we get contract ABI and bytecode, we could deploy it to Travis chain.

```
# Get Travis console
travis attach http://127.0.0.1:8545

# Deploy contract (in Travis console)
personal.unlockAccount(cmt.accounts[0], '1234');
abi = [{"constant":true,"inputs":[{"name":"src","type":"string"}],"name":"reverse",
↪"outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":"pure",
↪"type":"function"},{"constant":true,"inputs":[],"name":"test","outputs":[{"name":"",
↪"type":"string"}],"payable":false,"stateMutability":"pure","type":"function"}]
bytecode =
↪"0x608060405234801561001057600080fd5b5061041a806100206000396000f30060806040526004361061004c5760003
↪"
contract = web3.cmt.contract(abi);
c = contract.new(
  {
    from: web3.cmt.accounts[0],
    data: bytecode,
    gas: "4700000"
  },
  function(e, contract) {
    console.log("contract address: " + contract.address);
    console.log("transactionHash: " + contract.transactionHash);
  }
);
```

Use contract to reverse a string

```
# Use test() function to show reverse of "Hello, world"
> c.test.call();
"dlrow ,olleH"

# Use reverse() function to reverse a string
> c.reverse.call('abcdef');
"fedcba"
```

RSA Crypto

This example shows how to use our `eni_crypto.so` to do RSA encrypt / decrypt.

Compiler RSACrypto Contract

There is a simple contract that leverage ENI to do RSA encrypt / decrypt.

```
pragma lity ^1.2.4;

contract RSACrypto {
    function encrypt(string pubkey, string plaintext) public pure returns (string) {
        string memory ret;
        ret = eni("rsa_encrypt", pubkey, plaintext);
        return ret;
    }

```

(continues on next page)

(continued from previous page)

```
    function decrypt(string prikey, string ciphertext) public pure returns (string) {
        string memory ret;
        ret = eni("rsa_decrypt", prikey, ciphertext);
        return ret;
    }
}
```

and we could compile it using `lityc`:

```
$ mkdir output
$ lityc --abi --bin -o output RSACrypto.sol
$ cat output/RSACrypto.abi
[{"constant":true,"inputs":[{"name":"pubkey","type":"string"},{"name":"plaintext",
↪"type":"string"}],"name":"encrypt","outputs":[{"name":"","type":"string"}],"payable
↪":false,"stateMutability":"pure","type":"function"},{"constant":true,"inputs":[{
↪"name":"prikey","type":"string"},{"name":"ciphertext","type":"string"}],"name":
↪"decrypt","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":
↪"pure","type":"function"}]
$ cat output/RSACrypto.bin
608060405234801561001057600080fd5b506105e9806100206000396000f30060806040526004361061004c576000357c010
```

Deploy contract to Travis locally

After we get contract ABI and bytecode, we could deploy it to Travis chain.

```
# Get Travis console
travis attach http://127.0.0.1:8545

# Deploy contract (in Travis console)
personal.unlockAccount(cmt.accounts[0], '1234');
abi = [{"constant":true,"inputs":[{"name":"pubkey","type":"string"},{"name":"plaintext
↪","type":"string"}],"name":"encrypt","outputs":[{"name":"","type":"string"}],
↪"payable":false,"stateMutability":"pure","type":"function"},{"constant":true,"inputs
↪":[{"name":"prikey","type":"string"},{"name":"ciphertext","type":"string"}],"name":
↪"decrypt","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":
↪"pure","type":"function"}]
bytecode =
↪"0x608060405234801561001057600080fd5b506105e9806100206000396000f30060806040526004361061004c5760003
↪"
contract = web3.cmt.contract(abi);
c = contract.new(
  {
    from: web3.cmt.accounts[0],
    data: bytecode,
    gas: "4700000"
  },
  function(e, contract) {
    if (contract.address) {
      console.log("contract address: " + contract.address);
      console.log("transactionHash: " + contract.transactionHash);
    }
  }
);
```

Use contract to do RSA encrypt / decrypt

```
# Setup private & public keys
prikey = "-----BEGIN RSA PRIVATE KEY-----\nMIIEowIBAAKCAQEApr/
↪atzUuBArbsWHkn8tUmq00tCV4CcLfVUVg+zr41sixYIb5\n4rd5EFWoQ1xecYMZIbnoTl2vu9awHBZF18DkNlG8pjw1Vw5EjaW
↪7g\nGk0aWP/E1XoqkZJnBUYN5N6mOgtV3jv62w+XlNlozUySI0mBmjgyymAhPm4qx5Zc\nf/
↪Wmg42vbIFRrBl5PgWKGsY0L7xcPRDPAPRtndUPr+CLLk5KjyHI8a2WiYrJvjUG\nTQNyqPM5MmLLfHMkjkbE6DshWbMZona+/
↪5ji3wIDAQABAoIBAAjwNdAmSJ4s2tPq\nVHAAXTuhVzbk30deq8wNWQJ+icIxpdhvw8tUXGf0v31E4UciaOF27q3stbPS8UPA\
↪FXWck+IPThV56l+P\n4Hh82cgKglsKAUyBK7SWQiz0rpoj8MWlkG0TblsMVLnOTAO0N3p3NiHxv1eUJrHK\nwyI42Mkb+nUm1j
↪O8cRxSv6gU4bWH6YE24XQz/pRvOsLLcQeXrxbxvm1ZsD65ou\ntpvA0/
↪eF3c5KRAhoqgRGPDV7eHvRdo9v6Ih4mwp6wR9bEGU3beHCIjZPb5nCCGtk\nTCNiVt+MIXKBHXt9lKBjTnmbCvRt+chRz8yFwRp
↪+h65/\nAuvahIugw5AA+H8iTTeB2KpgCc2FmiUviohug39GMz6oabkzZH9KAZjCf5/
↪zMhm3\nIvtVDMDXBJah7SFYsxM1sBfklPAHFlAe7zP/950CgYBM60IZzonRPv/
↪0MKT18j97\n+PRibPHtsrywaQhzfhIpLsPek9gf5Vq4H5U40rkUoxtRWq6r7YJOZ7M44aWekicr\n4Ugvb8vKEdA9+T3yk9E2v
↪G2UDRuSpjcPuGuCOiIr1/
↪RmhmvRr+AerT\nz1jnCfdqNlYc14nQ4ajnswKBgDtlAj6lt25mePketwFbjpTOfkCLtI4Gfhrufaog\nJdNSXxa0paiYUfXadf
↪5N7pdG2m\nLWs9AoGBAMEgKXwA2ubWrxe622PHXwgUx9oja3LwmuT3oQZDtwxfs4lw3xzIgGps\nWVvgNL2aceE/
↪qkI032ysKTIbM3JvKa7AzrGKDi8XbyE98QSKM9qyFmdrTG7UIbSo\nDNen8V4qgCV/
↪z34+6uxWMR7AozgQmzrKogmxhZpIYdyqO4F35cMb\n-----END RSA PRIVATE KEY-----";
pubkey = "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEApr/
↪atzUuBArbsWHkn8tU\nmq00tCV4CcLfVUVg+zr41sixYIb54rd5EFWoQ1xecYMZIbnoTl2vu9awHBZF18Dk\nNlG8pjw1Vw5Eja
↪7gGk0aWP/E1XoqkZJnBUYN5N6mOgtV3jv62w+X\nlNlozUySI0mBmjgyymAhPm4qx5Zcf/
↪Wmg42vbIFRrBl5PgWKGsY0L7xcPRDPAPRt\nndUPr+CLLk5KjyHI8a2WiYrJvjUGTQNyqPM5MmLLfHMkjkbE6DshWbMZona+/
↪5ji\n3wIDAQAB\n-----END PUBLIC KEY-----";

# Encrypt
> ciphertext = c.encrypt.call(pubkey, 'Hello World!')

↪"49d511a44a3d2a2437f00a72fef6915fd0fcd669e90bd2f62387c6bdd3028ecc5b8799bae0a8bae606655046c87416297
↪"

# Decrypt
> c.decrypt.call(prikey, ciphertext)
"Hello World!"
```

Verify Dogecoin Block on Travis

Dogecoin verifier is a smart contract on Travis to verify correctness of Dogecoin block header and its pow hash. There is a step by step tutorial for running Dogecoin verifier on Travis locally.

Scrypt ENI Library

Dogecoin uses scrypt as its PoW hash algorithm. Our scrypt ENI library provides a simple interface for using scrypt:

- Input: `[0-9a-fA-F]{0,160}`, hex format of 0 to 80 bytes data
- Output: `[0-9a-f]{64}` hex format of 32 bytes data

Compile DogecoinVerifier Contract

There is a simple contract that leverage ENI to do heavy scrypt hash algorithm.

```
pragma lity ^1.2.4;

contract DogecoinVerifier {

  struct DogecoinBlockHeader {
    uint version;
    string prev_block;
    string merkle_root;
    uint timestamp;
    string bits;
    uint nonce;
  }

  function verifyBlock(uint version, string prev_block, string merkle_root, uint␣
↪timestamp, string bits, uint nonce) pure public returns (bool) {
    DogecoinBlockHeader memory block_header = DogecoinBlockHeader(version, prev_block,
↪ merkle_root, timestamp, bits, nonce);
    string memory block_header_hex = generateBlockHeader(block_header);
    string memory pow_hash = reverseHex(eni("scrypt", block_header_hex));
    uint256 target = bitsToTarget(bits);
    if (hexToUint(pow_hash) > target) {
      return false;
    }
    return true;
  }

  function generateBlockHeader(DogecoinBlockHeader header) pure internal returns␣
↪(string) {
    bytes memory block_header = new bytes(160);
    bytes memory version_hex = bytes(reverseHex(uintToHex(header.version, 8)));
    bytes memory prev_block_hex = bytes(reverseHex(header.prev_block));
    bytes memory merkle_root_hex = bytes(reverseHex(header.merkle_root));
    bytes memory timestamp_hex = bytes(reverseHex(uintToHex(header.timestamp, 8)));
    bytes memory bits_hex = bytes(reverseHex(header.bits));
    bytes memory nonce_hex = bytes(reverseHex(uintToHex(header.nonce, 8)));
    uint i;
    uint index = 0;
    for (i = 0; i < version_hex.length; i++) { block_header[index++] = version_hex[i];
↪ }
    for (i = 0; i < prev_block_hex.length; i++) { block_header[index++] = prev_block_
↪hex[i]; }
    for (i = 0; i < merkle_root_hex.length; i++) { block_header[index++] = merkle_
↪root_hex[i]; }
    for (i = 0; i < timestamp_hex.length; i++) { block_header[index++] = timestamp_
↪hex[i]; }
    for (i = 0; i < bits_hex.length; i++) { block_header[index++] = bits_hex[i]; }
    for (i = 0; i < nonce_hex.length; i++) { block_header[index++] = nonce_hex[i]; }
    return string(block_header);
  }

  function bitsToTarget(string bits) pure internal returns (uint) {
    uint bits_uint32 = hexToUint(bits);
    uint p = (bits_uint32 & 0xff000000) >> 24;
    p = (p - 3) * 8;
    uint result = (bits_uint32 & 0xffffff) << p;
    return result;
  }
```

(continues on next page)

(continued from previous page)

```
  function reverseHex(string hex_string) pure internal returns (string) {
    bytes memory hex_bytes = bytes(hex_string);
    assert(hex_bytes.length % 2 == 0);

    bytes memory outpute_bytes = new bytes(hex_bytes.length);
    for (uint i = 0; i < hex_bytes.length; i += 2) {
      outpute_bytes[i]     = hex_bytes[hex_bytes.length - i - 2];
      outpute_bytes[i + 1] = hex_bytes[hex_bytes.length - i - 1];
    }

    return string(outpute_bytes);
  }

  function hexToUint(string hex_string) pure internal returns (uint256) {
    bytes memory hex_bytes = bytes(hex_string);
    uint i;
    uint256 result = 0;
    for (i = 0; i < hex_bytes.length; i++) {
      result <<= 4;
      if (uint8(hex_bytes[i]) >= uint8(byte('0')) && uint8(hex_bytes[i]) <=␣
↪uint8(byte('9'))) {
        result += uint8(hex_bytes[i]) - uint8(byte('0'));
      } else {
        result += uint8(hex_bytes[i]) - uint8(byte('a')) + 10;
      }
    }

    return result;
  }

  function uintToHex(uint256 a, uint8 length) pure internal returns (string) {
    uint i;
    bytes memory hex_bytes = new bytes(64);
    for (i = 0; i < 64; i++) {
      byte last_half_byte = byte(a & 0xf);
      if (last_half_byte >= 0 && last_half_byte <= 9) {
        hex_bytes[63 - i] = byte(uint8(byte('0')) + uint8(last_half_byte));
      } else {
        hex_bytes[63 - i] = byte(uint8(byte('a')) + uint8(last_half_byte) - 10);
      }
      a >>= 4;
    }

    bytes memory output_bytes = new bytes(length);
    for (i = 0; i < length; i++) {
      output_bytes[i] = hex_bytes[64 - length + i];
    }
    return string(output_bytes);
  }
}
```

and we could compile it using `lityc`:

```
$ mkdir output
$ lityc --abi --bin -o output DogecoinVerifier.sol
$ cat output/DogecoinVerifier.abi
```

(continues on next page)

(continued from previous page)

```
[{"constant":true,"inputs":[{"name":"version","type":"uint256"},{"name":"prev_block",
↪"type":"string"},{"name":"merkle_root","type":"string"},{"name":"timestamp","type":
↪"uint256"},{"name":"bits","type":"string"},{"name":"nonce","type":"uint256"}],"name
↪":"verifyBlock","outputs":[{"name":"","type":"bool"}],"payable":false,
↪"stateMutability":"pure","type":"function"}]
$ cat output/DogecoinVerifier.bin
608060405234801561001057600080fd5b506111e380610020600039 6000f300608060405260043610610041576000357c01
```

Deploy contract to Travis locally

After we get contract ABI and bytecode, we could deploy it to Travis chain.

```
# Get Travis console
travis attach http://127.0.0.1:8545

# Deploy contract (in Travis console)
personal.unlockAccount(cmt.accounts[0], '1234');
abi = [{"constant":true,"inputs":[{"name":"version","type":"uint256"},{"name":"prev_
↪block","type":"string"},{"name":"merkle_root","type":"string"},{"name":"timestamp",
↪"type":"uint256"},{"name":"bits","type":"string"},{"name":"nonce","type":"uint256"}
↪],"name":"verifyBlock","outputs":[{"name":"","type":"bool"}],"payable":false,
↪"stateMutability":"pure","type":"function"}];
bytecode = "0x608060405234801561001057600080fd5b506111e380610020600039 6000f30060806040526004361061004
contract = web3.cmt.contract(abi);
c = contract.new(
  {
    from: web3.cmt.accounts[0],
    data: bytecode,
    gas: "4700000"
  },
  function(e, contract) {
    console.log("contract address: " + contract.address);
    console.log("transactionHash: " + contract.transactionHash);
  }
);
```

Use contract to verify pow hash

It’s time to verify block header using deployed contract. Here we use block #2 of dogecoin as an example.

To verify a block, we need the following block header information:

- version
- previous block hash
- transaction merkle root hash
- timestamp
- difficulty (bits)
- nonce

In block 2, these values in block header are:

- version: `1`

- previous block hash: `82bc68038f6034c0596b6e313729793a887fded6e92a31fbdf70863f89d9bea2`
- transaction merkle root hash: `3b14b76d22a3f2859d73316002bc1b9bfc7f37e2c3393be9b722b62bbd786983`
- timestamp: `1386474933` (convert from `2013-12-07 19:55:33 -0800`)
- difficulty (bits): `1e0ffff0`
- nonce: `3404207872`

Back to our DogecoinVerifier contract. The `verifyBlock` function use ENI library to compute scrypt hash of block header, and check the correctness of PoW result. The funcntion will return `true` if this block header is valid.

```
# Block #2 of dogecoin
> c.verifyBlock.call(1,
↪"82bc68038f6034c0596b6e313729793a887fded6e92a31fbdf70863f89d9bea2",
↪"3b14b76d22a3f2859d73316002bc1b9bfc7f37e2c3393be9b722b62bbd786983", 1386474933,
↪"1e0ffff0", 3404207872)
true

# Even 1-bit of nonce changed, this block header will become invalid
> c.verifyBlock.call(1,
↪"82bc68038f6034c0596b6e313729793a887fded6e92a31fbdf70863f89d9bea2",
↪"3b14b76d22a3f2859d73316002bc1b9bfc7f37e2c3393be9b722b62bbd786983", 1386474933,
↪"1e0ffff0", 3404207871)
false

# You could also use sendTransaction to verifyBlock function
> t = c.verifyBlock.sendTransaction(2,
↪"12aca0938fe1fb786c9e0e4375900e8333123de75e240abd3337d1b411d14ebe",
↪"31757c266102d1bee62ef2ff8438663107d64bdd5d9d9173421ec25fb2a814de", 1392346781,
↪"1b267eeb", 2216773632, {from:web3.cmt.accounts[0], gas: 1000000});

# We set the gas limit to 1000000 and verifyBlock function only costs 243583 gas
> cmt.getTransaction(t)
{
  blockHash: "0x46fc1763dd3c761e84c6f0a6eb430333c0eb8b3b33d8d2d6098d48b3b7662459",
  blockNumber: 39,
  from: "0x7eff122b94897ea5b0e2a9abf47b86337fafebdc",
  gas: 1000000,
  gasPrice: 2000000000,
  hash: "0xa401b964930848e44af24a31338a58ca1f75e32a0abbb0f978ad3c519595e04a",
  input:
↪"0xa83dc30600000000000000000000000000000000000000000000000000000000000000020000000000000000000000
↪",
  nonce: 1,
  r: "0x62d1d11973bfd94f05daa86708806d23cadbb1f33b6d7b5cc1e2bec90d50907",
  s: "0x26df72607d09561de0d2507a8d5789676ddc4ce93ee3ac67f976ee64a61911c5",
  to: "0xb6b29ef90120bec597939e0eda6b8a9164f75deb",
  transactionIndex: 0,
  v: "0x4a",
  value: 0
}
> cmt.getTransactionReceipt(t)
{
  blockHash: "0x46fc1763dd3c761e84c6f0a6eb430333c0eb8b3b33d8d2d6098d48b3b7662459",
  blockNumber: 39,
  contractAddress: null,
  cumulativeGasUsed: 243583,
  from: "0x7eff122b94897ea5b0e2a9abf47b86337fafebdc",
```

(continues on next page)

(continued from previous page)

```
  gasUsed: 243583,
  logs: [],
  logsBloom:
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
  root: "0xaa9759a788044dbad69ae99e521c1c5969e0b4e52e17284313ce367f3df48f9e",
  to: "0xb6b29ef90120bec597939e0eda6b8a9164f75deb",
  transactionHash: "0xa401b964930848e44af24a31338a58ca1f75e32a0abbb0f978ad3c519595e04a
↪",
  transactionIndex: 0
}
```

2.4.3 Other documentations

Validator Only Contract

Warning: Lity supports Validator Only Contracts after lity v1.2.2.

Lity supports Validator Only Contracts after v1.2.2. By providing the built-in function called *isValidator* and special opcode *OpIsValidator*, Lity can check whether an address is a validator of CyberMiles blockchain. With this feature, smart contract developers can create trusted smart contracts that can only be modified by the validators.

A common use case is to serve as oracles to provide trusted states.

isValidator Grammar

```
isValidator(<address>) returns (bool returnValue);

// isValidator is a built-in function provided by Lity.
// isValidator only takes one parameter, an address, to check whether this address is␣
↪a validtor.
// If the address is a validator => return true
// Otherwise => return false
```

Examples

BTCRelay

Here we use BTCRelay contract as an example to show how validator only contract works. BTCRelay is a contract that store BTC headers and verify BTC transactions.

Download and compile it using `lityc` (version >= 1.2.2):

```
$ wget https://raw.githubusercontent.com/CyberMiles/smart_contracts/master/BTCRelay/
↪BTCRelay.sol
$ lityc --abi --bin -o output BTCRelay.sol
```

and then we could deploy it to Travis chain:

```
$ travis attach http://localhost:8545

// Prepare accounts

> validatorAccount = web3.cmt.accounts[0]
"0x7eff122b94897ea5b0e2a9abf47b86337fafebdc"
> normalAccount = personal.newAccount('test')
"0x34bb135d77771038a2fda3aa7cd5dc3fb4cc6abd"
> personal.unlockAccount(validatorAccount, '1234', 86400)
true
> personal.unlockAccount(normalAccount, 'test', 86400)
true
> cmt.sendTransaction({from: validatorAccount, to: normalAccount, value:␣
↪1000000000000000000})

// Deploy contract

> abi = <...>
> bytecode = <...>
> contract = web3.cmt.contract(abi);
> btcrelay = contract.new(
  {
    from: web3.cmt.accounts[0],
    data: bytecode,
    gas: "4700000"
  },
  function(e, contract) {
    if (typeof contract.address !== 'undefined') {
      console.log('Contract mined! address: ' + contract.address + ' transactionHash:
↪' + contract.transactionHash);
    }
  }
);

// Contract mined! address: 0x05ea0f73204ea39d7231706a49c174a20380cfec␣
↪transactionHash: 0x1acbd651d91f1edd3520da59dfda077c9f2ca3bbd9aa09bbc3304ec76a8b41ef
```

Now we try to call `storeHeader` function using both `validatorAccount` and `normalAccount`

```
> height = 125552
> header =
↪'0x0x0100000081cd02ab7e569e8bcd9317e2fe99f2de44d49ab2b8851ba4a308000000000000e320b6c2fffc8d750423dk
↪'

// Send transaction with validator account

> tx = btcrelay.storeHeader.sendTransaction(header, height, {from: validatorAccount,␣
↪gas: "300000"})
"0xe5d83d5ca82a00b6311ec9b8c0bab4df3f7a62971231791182686ba8f594b7eb"
> cmt.getTransactionReceipt(tx)
{
  blockHash: "0x91186a59e8fb19c1db440ab0697266fb3407a10844342278a55e1dcf0eb501e1",
  blockNumber: 72145,
  contractAddress: null,
  cumulativeGasUsed: 140024,
  from: "0x7eff122b94897ea5b0e2a9abf47b86337fafebdc",
  gasUsed: 140024,
```

(continues on next page)

(continued from previous page)

```
  logs: [{
      address: "0x05ea0f73204ea39d7231706a49c174a20380cfec",
      blockHash: "0x91186a59e8fb19c1db440ab0697266fb3407a10844342278a55e1dcf0eb501e1",
      blockNumber: 72145,
      data: "0x0000000000000001e8d6829a8a21adc5d38d0a473b144b6765798e61f98bd1d",
      logIndex: 0,
      removed: false,
      topics: ["0xf82c50f1848136e6c140b186ea0c768b7deda5efffe42c25e96336a90b26c744"],
      transactionHash:
↪"0xe5d83d5ca82a00b6311ec9b8c0bab4df3f7a62971231791182686ba8f594b7eb",
      transactionIndex: 0
  }],
  logsBloom:
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000100000000000000010002000000000000000000000000000000000
0000000000000000000000000000000000000000000004000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000",
  status: "0x1",
  to: "0x05ea0f73204ea39d7231706a49c174a20380cfec",
  transactionHash: "0xe5d83d5ca82a00b6311ec9b8c0bab4df3f7a62971231791182686ba8f594b7eb
↪",
  transactionIndex: 0
}

// Send transaction with normal account

> tx = btcrelay.storeHeader.sendTransaction(header, height, {from: normalAccount,␣
↪gas: "300000"})
"0xed05e610feae32bd51931b9d0068104dc60a9595590d17b73e1916c2288b0cf9"
> cmt.getTransactionReceipt(tx)
{
  blockHash: "0x611671eff48e414706daa66f1fcc404428832dd79c2b6a9481a28f46ef93a430",
  blockNumber: 72173,
  contractAddress: null,
  cumulativeGasUsed: 27466,
  from: "0x34bb135d77771038a2fda3aa7cd5dc3fb4cc6abd",
  gasUsed: 27466,
  logs: [],
  logsBloom:
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000",
  status: "0x0",
  to: "0x05ea0f73204ea39d7231706a49c174a20380cfec",
  transactionHash: "0xed05e610feae32bd51931b9d0068104dc60a9595590d17b73e1916c2288b0cf9
↪",
  transactionIndex: 0
}
```

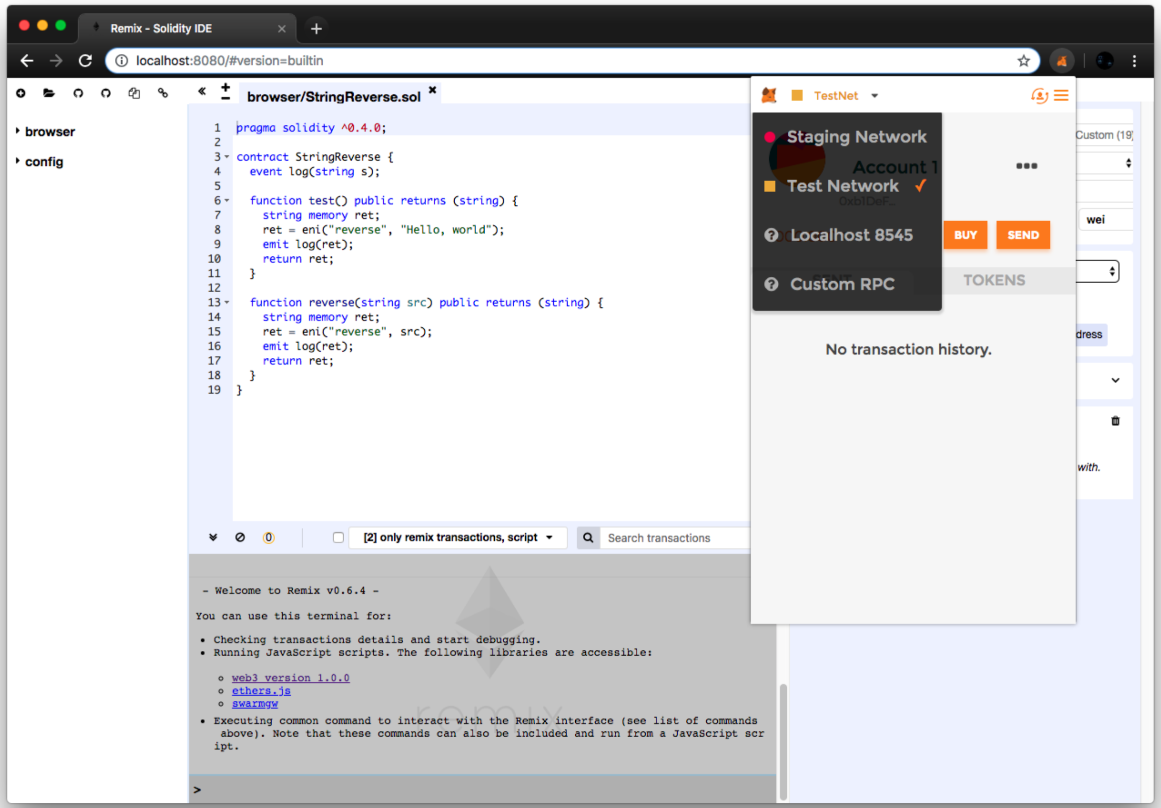
We could see when sending transaction using `validatorAccount`, the transactions triggered `storeHeader` function successfully and emit log. Log data `0x0000000000000001e8d6829a8a21adc5d38d0a473b144b6765798e61f98bd1d` is returned hash of block header. When sending transaction using `normalAccount`, it could not pass `validatorOnly()` check and failed.

If you want to interact with BTCRelay contract, you could check our fetchd as an example to fetch and store BTC headers.

Integrate CyberMiles Remix & MetaMask

Install CyberMiles MetaMask Chrome extension

- Download chrome.crx from https://www.cybermiles.io/metamask/
- Install chrome.crx (Go to chrome://extensions/ and drag the chrome.crx extension file to extension page)
- Open MetaMask to initialize CMT account
- Switch to **Test Network**



```
pragma solidity ^0.4.0;

contract StringReverse {
  event log(string s);

  function test() public returns (string) {
    string memory ret;
    ret = eni("reverse", "Hello, world");
    emit log(ret);
    return ret;
  }

  function reverse(string src) public returns (string) {
    string memory ret;
    ret = eni("reverse", src);
    emit log(ret);
    return ret;
  }
}
```

```
- Welcome to Remix v0.6.4 -

You can use this terminal for:

• Checking transactions details and start debugging.
• Running JavaScript scripts. The following libraries are accessible:

  ○ web3 version 1.0.0
  ○ ethers.js
  ○ swarmgw
• Executing common command to interact with the Remix interface (see list of commands
  above). Note that these commands can also be included and run from a JavaScript scr
  ipt.

>
```

- Get some CMT at http://travis-faucet.cybermiles.io/cn/

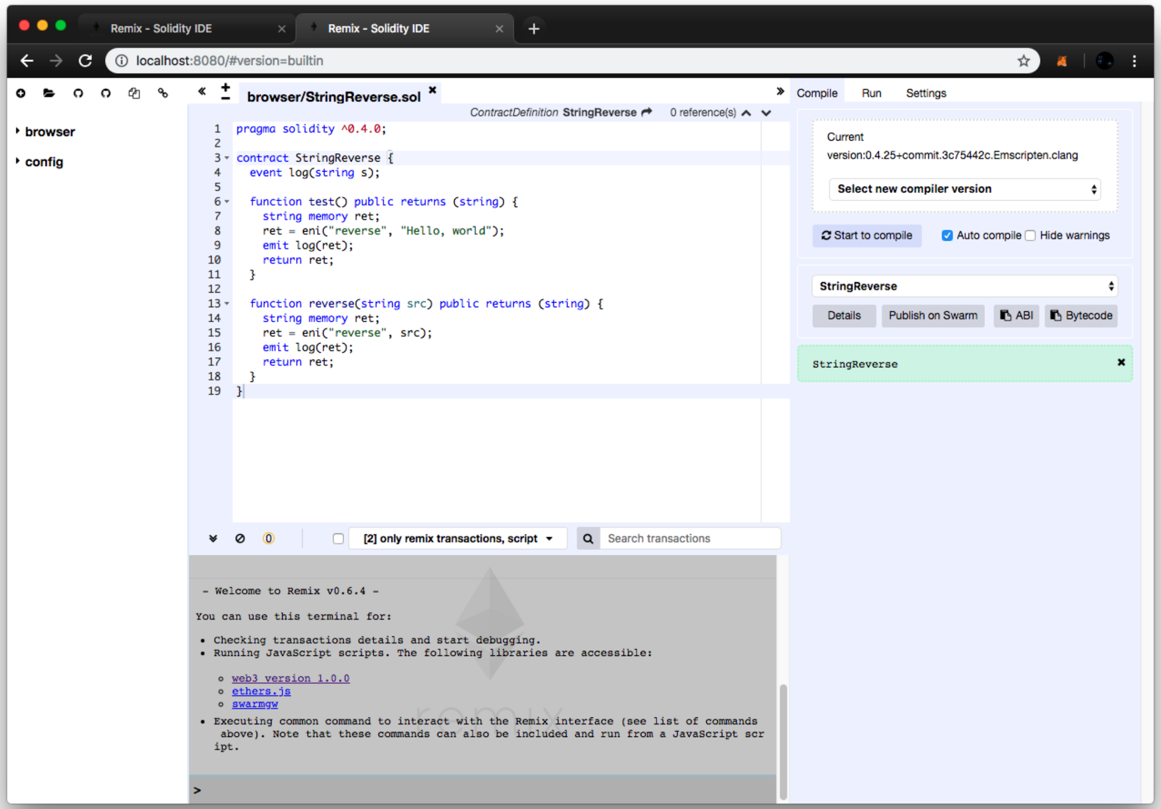
Run Remix IDE

```
git clone https://github.com/CyberMiles/remix-ide.git
cd remix-ide
npm install
npm run setupremix
npm start
```

- It will run Remix IDE on default port 8080

Using MetaMask in Remix IDE

- Check Remix IDE at http://localhost:8080
- Turn on *Auto compile* option



Current
version:0.4.25+commit.3c75442c.Emscripten.clang

```
pragma solidity ^0.4.0;

contract StringReverse {
  event log(string s);

  function test() public returns (string) {
    string memory ret;
    ret = eni("reverse", "Hello, world");
    emit log(ret);
    return ret;
  }

  function reverse(string src) public returns (string) {
    string memory ret;
    ret = eni("reverse", src);
    emit log(ret);
    return ret;
  }
}
```

```
- Welcome to Remix v0.6.4 -

You can use this terminal for:

• Checking transactions details and start debugging.
• Running JavaScript scripts. The following libraries are accessible:

  o web3 version 1.0.0
  o ethers.js
  o swarmgw

• Executing common command to interact with the Remix interface (see list of commands
  above). Note that these commands can also be included and run from a JavaScript scr
  ipt.

>
```

- Go to run tab at right panel and select *Injected Web3* as Environment)
- Use *StringReverse.sol* as example

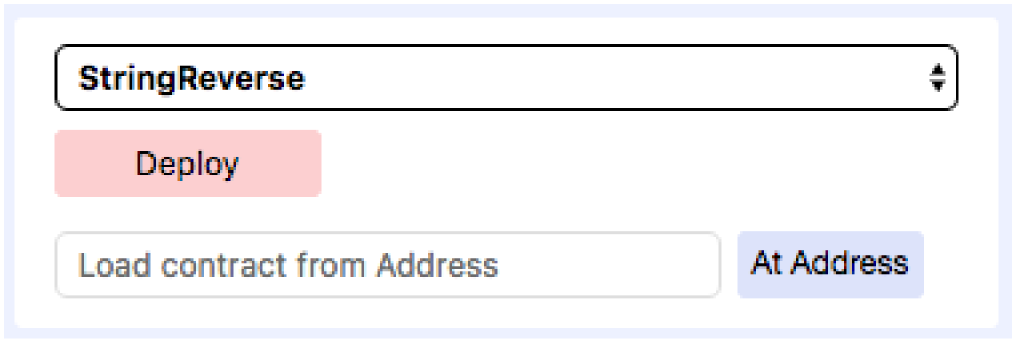
```
pragma lity ^1.2.4;

contract StringReverse {
  event log(string s);

  function test() public returns (string) {
    string memory ret;
    ret = eni("reverse", "Hello, world");
    emit log(ret);
    return ret;
  }

  function reverse(string src) public returns (string) {
    string memory ret;
    ret = eni("reverse", src);
    emit log(ret);
    return ret;
```

(continues on next page)

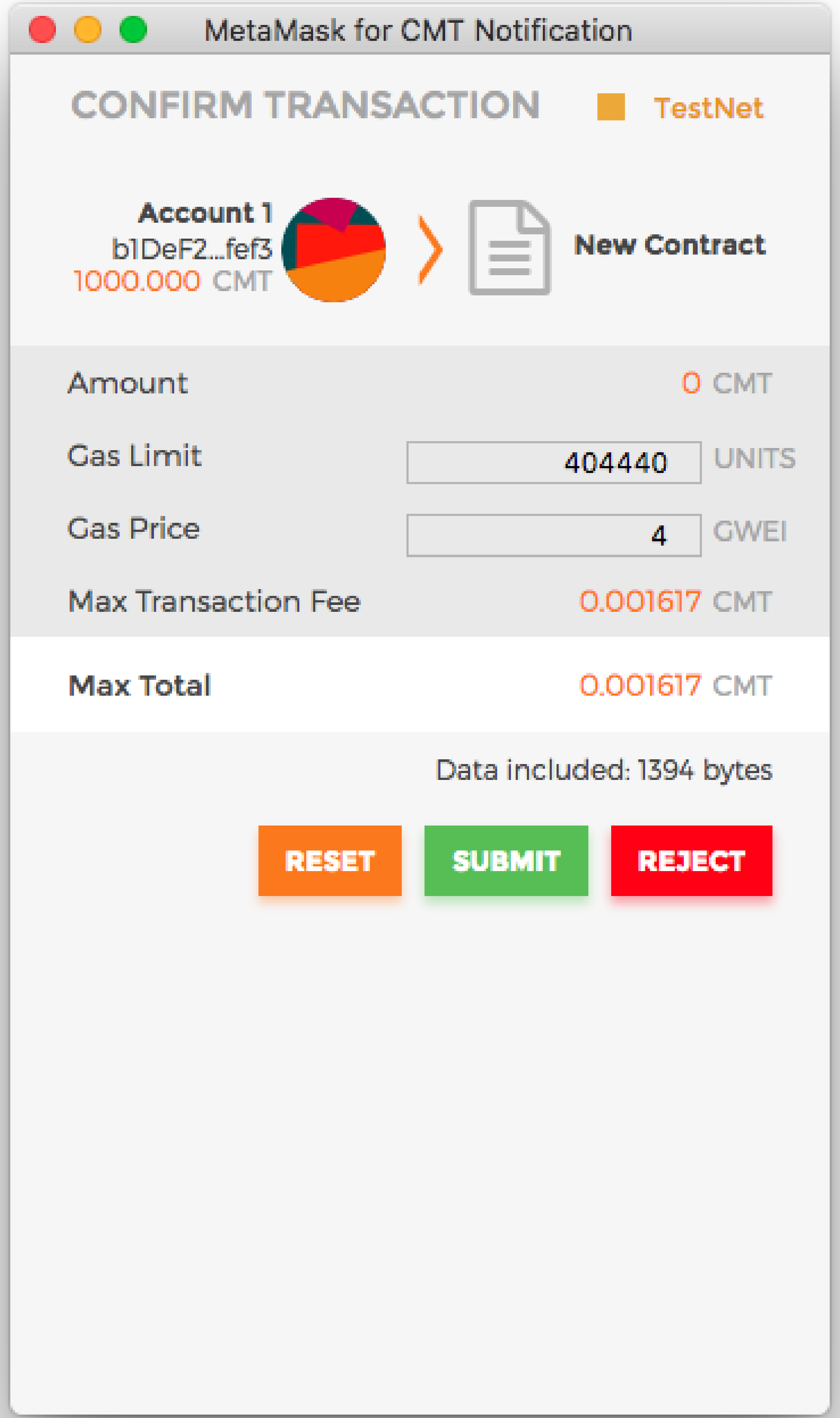
(continued from previous page)

```
    }
}
```

- Click red *Deploy* button to deploy contract



- Submit create contract transaction in MetaMask



CONFIRM TRANSACTION TestNet

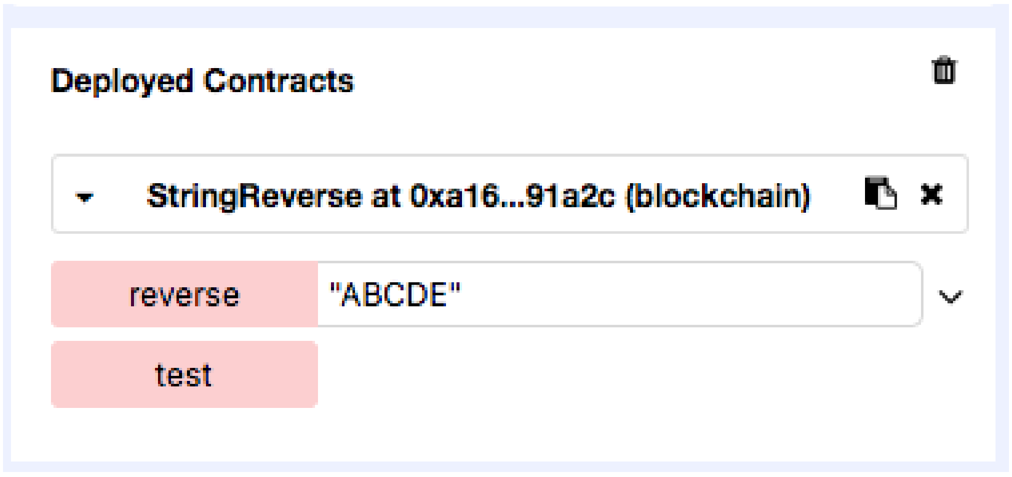
Account 1
b1DeF2...fef3
1000.000 CMT

New Contract

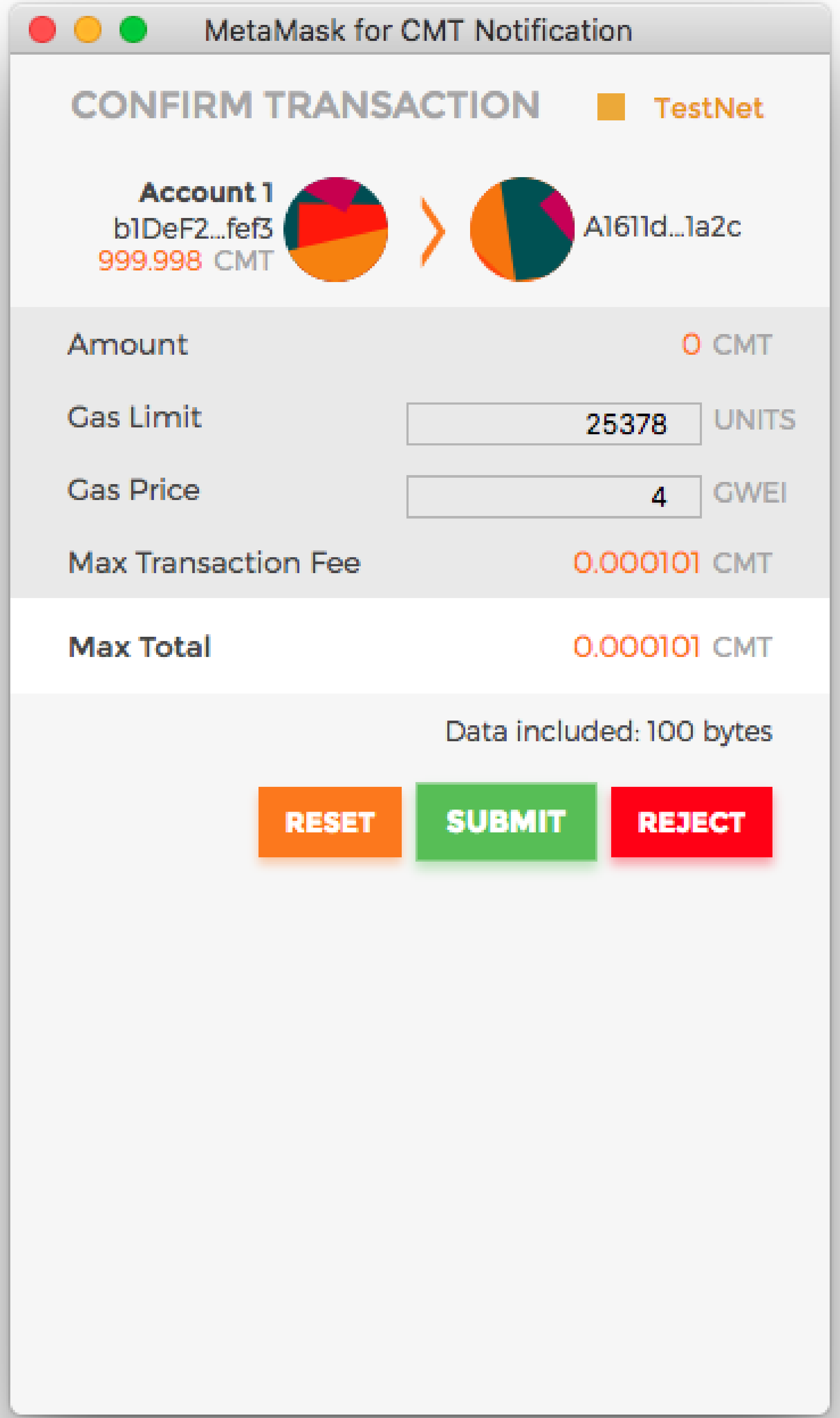
Amount	0 CMT
Gas Limit	404440 UNITS
Gas Price	4 GWEI
Max Transaction Fee	0.001617 CMT
Max Total	0.001617 CMT

Data included: 1394 bytes

- Now we try to submit *reverse("ABCDE")* function call (click red *reverse* button to submit)



- Submit reverse call transaction in MetaMask

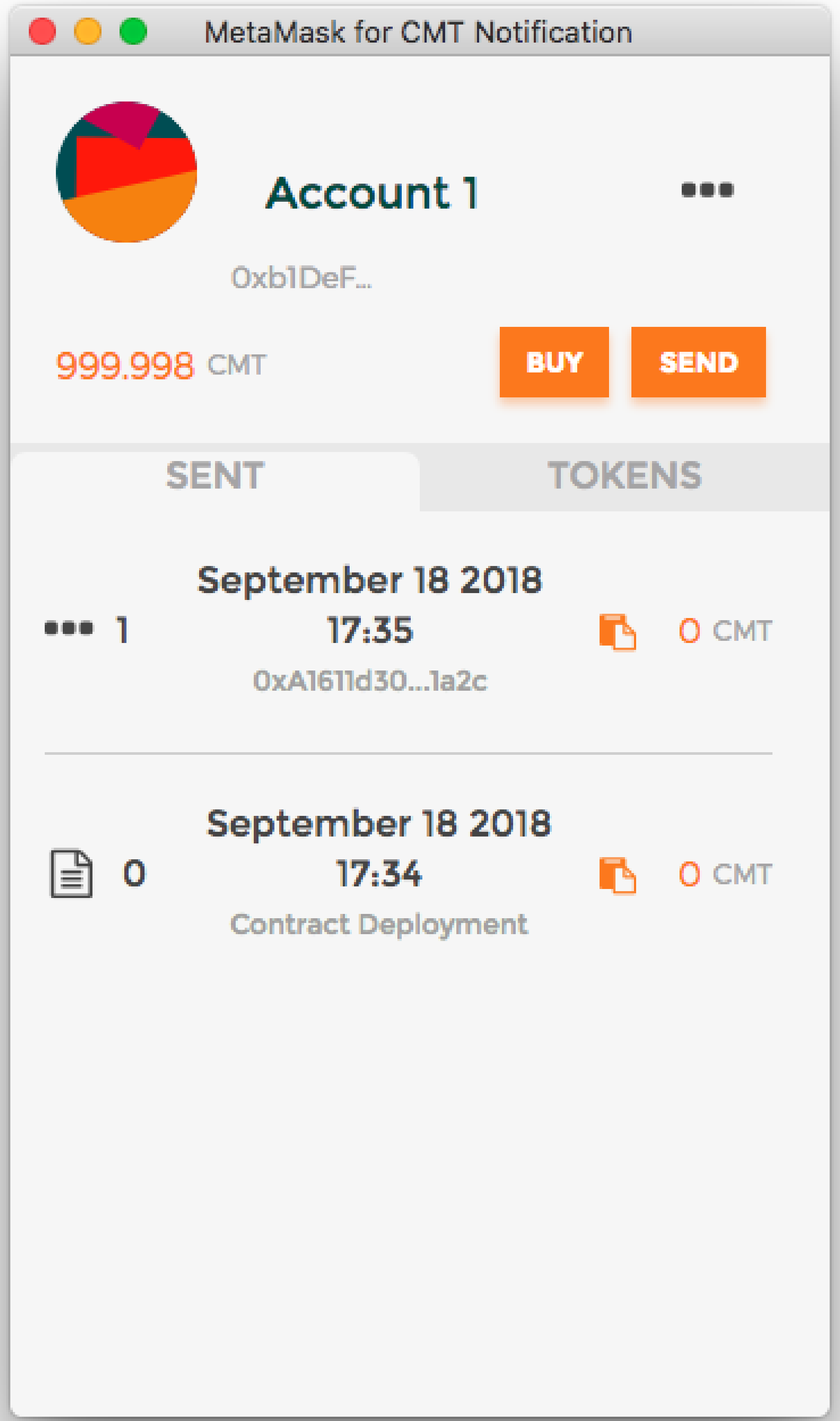


Account 1
b1DeF2...fef3
999.998 CMT

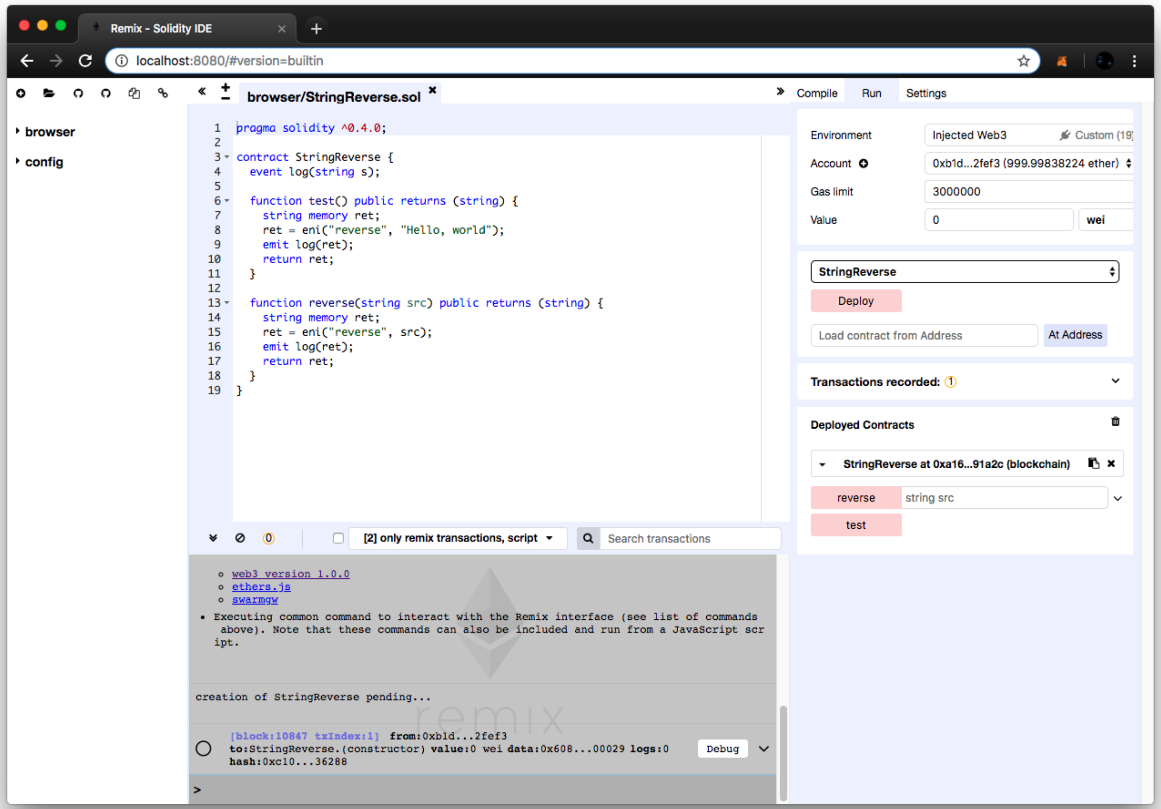
A1611d...1a2c

Amount	0 CMT
Gas Limit	25378 UNITS
Gas Price	4 GWEI
Max Transaction Fee	0.000101 CMT
Max Total	0.000101 CMT

Data included: 100 bytes


MetaMask for CMT Notification
Account 1
0xb1DeF...
999.998 CMT
BUY
SEND
SENT
TOKENS
September 18 2018
17:35
1
0xA1611d30...1a2c
0 CMT
September 18 2018
17:34
0
Contract Deployment
0 CMT

- Check result in Remix IDE console



```
pragma solidity ^0.4.0;

contract StringReverse {
  event log(string s);

  function test() public returns (string) {
    string memory ret;
    ret = eni("reverse", "Hello, world");
    emit log(ret);
    return ret;
  }

  function reverse(string src) public returns (string) {
    string memory ret;
    ret = eni("reverse", src);
    emit log(ret);
    return ret;
  }
}
```

Environment: Injected Web3 — Custom (19

Account: 0xb1d...2fef3 (999.99838224 ether)

Gas limit: 3000000

Value: 0 wei

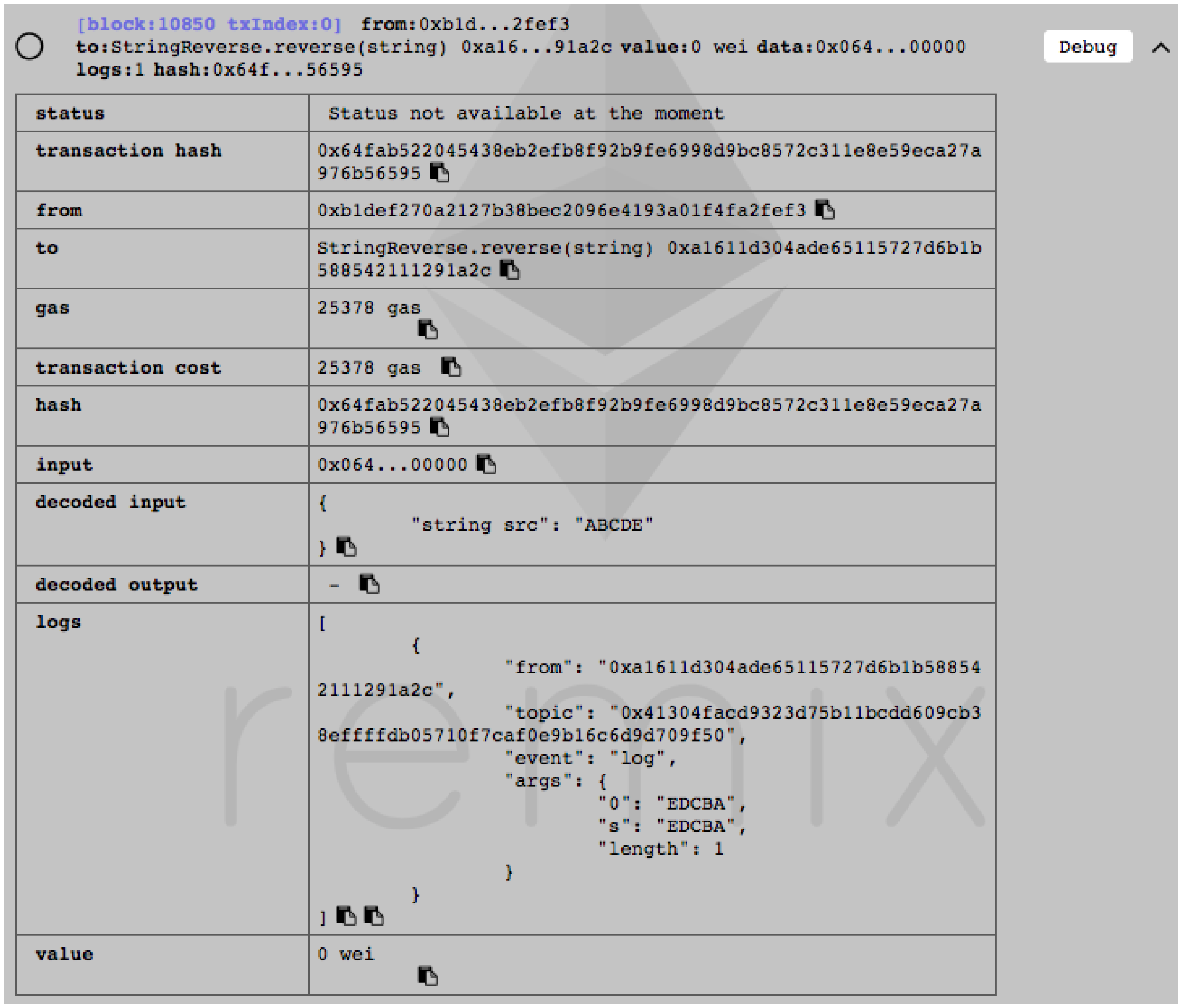
StringReverse

Transactions recorded: 1

Deployed Contracts

StringReverse at 0xa16...91a2c (blockchain)

```
  o web3 version 1.0.0
  o ethers.js
  o swarmgw
• Executing common command to interact with the Remix interface (see list of commands
   above). Note that these commands can also be included and run from a JavaScript scr
  ipt.

creation of StringReverse pending...

    [block:10847 txIndex:1] from:0xb1d...2fef3
    to:StringReverse.(constructor) value:0 wei data:0x608...00029 logs:0
    hash:0xc10...36288

>
```



[block:10850 txIndex:0] from:0xb1d...2fef3 to:StringReverse.reverse(string) 0xa16...91a2c value:0 wei data:0x064...00000 logs:1 hash:0x64f...56595

Debug

status	Status not available at the moment
transaction hash	0x64fab522045438eb2efb8f92b9fe6998d9bc8572c311e8e59eca27a976b56595
from	0xb1def270a2127b38bec2096e4193a01f4fa2fef3
to	StringReverse.reverse(string) 0xa1611d304ade65115727d6b1b588542111291a2c
gas	25378 gas
transaction cost	25378 gas
hash	0x64fab522045438eb2efb8f92b9fe6998d9bc8572c311e8e59eca27a976b56595
input	0x064...00000
decoded input	{ "string src": "ABCDE" }
decoded output	-
logs	[{ "from": "0xa1611d304ade65115727d6b1b588542111291a2c", "topic": "0x41304facd9323d75b11bcdd609cb38efffdb05710f7caf0e9b16c6d9d709f50", "event": "log", "args": { "0": "EDCBA", "s": "EDCBA", "length": 1 } }]
value	0 wei

Oyente Integration

Lity now integrate analysis tool Oyente and generates analysis report after compiling contract if Oyente installed.

Requirements

Install our lityc and evm.

Installation

Execute a python virtualenv

```
virtualenv -p python3 env
source env/bin/activate
```

Download Oyente

```
git clone https://github.com/CyberMiles/oyente.git
cd oyente
```

Install Oyente requirements

```
pip install -r requirements.txt
```

Install Oyente

```
pip install .
```

Usage

After running `lityc`, it will automatically invoke `oyente` command if installed.

```
$ cat StringReverse.sol

pragma lity ^1.2.4;
contract StringReverse {
  function test() public pure returns (string) {
    string memory ret;
    ret = eni("reverse", "Hello, world");
    return ret;
  }
  function reverse(string src) public pure returns (string) {
    string memory ret;
    ret = eni("reverse", src);
    return ret;
  }
}
```

```
$ lityc --abi StringReverse.sol

======= StringReverse.sol:StringReverse =======
Contract JSON ABI
[{"constant":true,"inputs":[{"name":"src","type":"string"}],"name":"reverse","outputs
↪":[{"name":"","type":"string"}],"payable":false,"stateMutability":"pure","type":
↪"function"},{"constant":true,"inputs":[],"name":"test","outputs":[{"name":"","type":
↪"string"}],"payable":false,"stateMutability":"pure","type":"function"}]

INFO:root:contract StringReverse.sol:StringReverse:
INFO:oyente.symExec:    ============ Results ===========
INFO:oyente.symExec:      EVM Code Coverage:                     29.8%
INFO:oyente.symExec:      Parity Multisig Bug 2:                 False
INFO:oyente.symExec:      Callstack Depth Attack Vulnerability:  False
INFO:oyente.symExec:      Transaction-Ordering Dependence (TOD): False
INFO:oyente.symExec:      Timestamp Dependency:                  False
INFO:oyente.symExec:      Re-Entrancy Vulnerability:             False
INFO:oyente.symExec:    ====== Analysis Completed ======
```

Rand - Get random number on the chain

Warning: Lity supports built-in random function after lity v1.2.7.

Built-in Random Function

Lity provides a new built-in function *rand* for the gaming Dapp developers.

The function prototype is *function rand() returns (uint)*. When calling *rand()*, developers will receive an random number which is generated by the following formula:

```
function rand() returns (uint) {
  return keccak256(Randomness Seed, Nonce, CodeHash, Counter);
}
```

- The *Randomness Seed* is generated by Travis chain and you can find it in the difficulty field of current block header.
- The *Nonce* is nonce of the transaction origin and depends on the transaction order in current block.
- The *CodeHash* is the hash of caller's contract bytecode
- The *Counter* will automaticlly increase after the *opRand* is called.

The developers should be aware of the usage of random number. See *getPureRandnum()* in following example. The return value of *getPureRandnum()* may be the same, because of the following limitation:

1. *Randomness Seed* depends on current block. It will be the same until receiving the next block.
2. *Nonce* depends on the transaction order. When you test in local environment and don;t produce a new transaction, you might get the same nonce.
3. *CodeHash* will not change after you deployed the contract.
4. *Counter* depends on the the called time of rand(). If you call *rand()* twice in same code block, you will receive different results. On the other hands, if you call *getPureRandnum()*, the *rand()* is called on different code block, you may receive the same result.

Example 1

```
pragma lity ^1.2.7;

contract RandNumberExample {
  uint randnum;
  function getAndSetRandnum() public returns (uint) {
    randnum = rand();
    return randnum;
  }
  function getPureRandnum() public pure returns (uint) {
    return rand();
  }
}
```

FreeGas - Let contract owner pay the tx fee for you

Warning: Lity supports built-in random function after lity v1.2.7.

Built-it FreeGas Modifier

Lity provides a new modifier *freegas*. With *freegas*, developers can mark specific functions to be *freegas*, which means the transaction fee will be paid by the callee contract and caller user doesn't need to care about the gas fee.

Because *freegas* will consume the contract balance for the transaction fee, Dapp developers should make sure that the deployed contract must have a fallback payable function and should send some money into the contract.

How Virtual Machine handles the freegas function

When `freegas` is triggered, VM will follow these rules to calculate transaction fee:

if Balance(Callee contract) >= transaction fee,

then Use Balance(Calle contract) to pay the fee and refund all of the reserved balance

else

then Use origin tx gas to pay the fee and refund the remaining balance

Example 1

```
pragma lity ^1.2.7;

contract FreeGas {
    uint x;
    function c(uint a) public freegas returns (uint) {
        x = a;
        return x;
    }
    function () public payable {}
}
```

Schedule Transaction

Warning: Schedule Transaction is under development.

Schedule Transaction is not fully supported by Lity yet.

Draft Grammar

```
schedule(<External Function Call>, <Timestamp>);

// <External Function Call>:
//     Some examples:
//       this.deposit(1000, 0x95...185)
```

(continues on next page)

(continued from previous page)

```
//      erc223receiver.tokenFallBack(...)
// <Timestamp>:
//    Unix timestamp
// The <External Function Call> will be executed at <Timestamp>.
```

Examples

```
// pragma lity ^1.3.0;

contract Schedule {
  function deposit(uint amount, address to) private {
      to.transfer(amount);
  }
  function setTimer(uint amount, address to, uint timestamp) public {
      schedule(this.deposit(amount, to), timestamp);
  }
}
```

Types

Fixed Point Numbers

Warning: Lity supports fixed point numbers after lity v1.2.6.

Keywords are `fixedMxN` for signed fixed point numbers, and `ufixedMxN` for unsigned fixed point numbers. `M` is the number of bits used by the type, and `N` is the number of fractional points. `M` should be a number in `[8, 256]` which is divisible by 8, and `N` is a positive integer that fits in an `uint32`.

`ufixed` and `fixed` are aliases to `ufixed128x18` and `fixed128x18` respectively.

Grammar

```
SignedFixed   = 'fixed' ( [0-9]+ 'x' [0-9]+ )?
UnsignedFixed = 'ufixed' ( [0-9]+ 'x' [0-9]+ )?
```

Operators

- Arithmetics: binary +, binary −, unary +, unary −, `*`, `/`, `%`
- Comparisons: <=, <, ==, !=, >=, >
- Conversions: see *Conversions between Literals and Fixed Point Types*.

Arithmetic operators may cause implicit *truncation*.

Definition

Take a look at Wikipedia's explanation for the definition of fixed point numbers.

You may check the ranges of some fixed point types listed below to better understand the concept of fixed point numbers.

Type	Range
ufixed8x1	[0.0, 25.5]
ufixed8x8	[0.00000000, 0.00000255]
fixed16x2	[-327.68, 32767]
fixed16x7	[-0.0032768, 0.0032767]

Conversions between Literals and Fixed Point Types

Decimal literals can be converted to the target fixed point type only if it fits into N bits.

Implcit Conversion

Decimal literals can be converted to fixed point types implicitly if no *truncation* occurred.

```
ufixed8x2 a = 1.1; // rational 11/10 to ufixed8x2, ok
fixed8x1 sa = 1.1; // rational 11/10 to fixed8x1, ok

ufixed8x2 b = 9.9; // rational 99/10 to ufixed8x2 [0.00, 2.55], fail
```

Explcit Conversion

When *truncation* will occur, explicit conversion is required.

```
ufixed16x2 pi = ufixed16x2(3.1415926535); // truncated to 3.14
```

Not Convertible

A TypeError is raised when the literal does not fit into N bits even if using explcit conversion.

```
// store 2.56 into ufixed8x1 [0.0, 25.5] results in TypeError
ufixed8x1 a = 25.6;
ufixed8x1 b = ufixed8x1(25.6);
```

Conversions between Different Fixed Point Types

Explicit conversion is also required between different fixed point types if it might cause *truncation*.

```
ufixed16x2 pi2 = 3.14;
ufixed16x4 pie = ufixed16x4(pi2); // still 3.14 but need explicit conversion
```

Warning: Conversions that causes overflow or underflow of fixed point types is currently undefined.

```
fixed8x1 a = -1.0;
fixed8x1 b = -0.1;
ufixed8x1 ua = ufixed8x1(a); // undefined
ufixed8x1 ub = ufixed8x1(b); // undefined
```

Truncations

If a fixed point number has more than `N` digits after its decimal point, all digits after the `N`-th digit are truncated.

Todo: Add some examples here.

ABI Types

`fixedMxN` for signed fixed point numbers, and `ufixedMxN` for unsigned fixed point numbers. The definition of `M` and `N` are the same as those in the definition of Lity fixed point numbers.

Compatibility with Solidity

Fixed point numbers are not fully supported in Solidity yet. The range of fractional points `N` in Solidity is currently `[0, 80]` (see Solidity's documentation for fixed point numbers). There's a discussion to change it to `[0, 77]` here at #4061.

VM Wrapper

VM Wrapper give a way to dynamic load evm shared library which compatible EVMC 5.0.0

Required

- >= EVMC 5.0

Example

Use Hera VM for example. First, clone it from github and build with *-DBUILD_SHARED_LIBS=ON*.

To demo the result, we can modify *hera.cpp* to show some information. Modified hera will show

- EVM binary code in hex.
- OP TIMESTAMP result.

```
diff --git a/src/hera.cpp b/src/hera.cpp
index 6a7d742..fd60a93 100644
--- a/src/hera.cpp
+++ b/src/hera.cpp
```

(continues on next page)

(continued from previous page)

```
@@ -363,13 +363,24 @@ evmc_result hera_execute(
   memset(&ret, 0, sizeof(evmc_result));

   try {
-    heraAssert(rev == EVMC_BYZANTIUM, "Only Byzantium supported.");
+    heraAssert(rev != EVMC_BYZANTIUM, "Only Byzantium supported.");
     heraAssert(msg->gas >= 0, "Negative startgas?");

     bool meterInterfaceGas = true;

     // the bytecode residing in the state - this will be used by interface methods␣
↪(i.e. codecopy)
     vector<uint8_t> state_code(code, code + code_size);
+
+    cout<<" Show Byte Code:"<<endl;
+    cout<<"    = size:"<<code_size<<endl;
+    for(int v:state_code)
+      cout<<hex<<v<<' ';
+    cout<<endl;
+
+    evmc_tx_context tx_context;
+    context->fn_table->get_tx_context(&tx_context, context);
+    time_t timestamp = tx_context.block_timestamp;
+    cout<< "TIMESTAMP: "<< put_time(localtime(&timestamp), "%c %Z") <<endl;

     // the actual executable code - this can be modified (metered or evm2wasm␣
↪compiled)
     vector<uint8_t> run_code(code, code + code_size);
```

Build Hera and copy srclibhera.so to where you want to.

```
git clone https://github.com/ewasm/hera.git --recuresive
cd hera
cp ../modified_hera.cpp ./src/hera.cpp
mkdir build
cd build
cmake .. -DBUILD_SHARED_LIBS=ON
make
```

Use EVM Wrapper

Go wrapper will check environment variable *EVMC_PATH* and load VM by this variable.

```
echo 64acacacacac7f7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff0c␣
↪> tmp.bin

export EVMC_PATH="./libhera.so"
./evm --codefile tmp.bin --debug run
```

Evm will show

```
Show Byte Code:
 = size:40
64 ac ac ac ac ac 7f 7f ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff␣
↪ff ff ff ff ff ff ff ff ff ff ff c
```

(continues on next page)

(continued from previous page)

```
TIMESTAMP: Sun Aug  5 01:41:31 2018 DST
0x
error: evmc: failure
#### TRACE ####
#### LOGS ####
```

ERC contract checker guide

Please make sure that lityc has been installed on your system. If not, please follow *Getting started*.

ERC20 Contract Standard Checker

Enable contract standard checker with specific ERC

```
$ lityc --contract-standard ERC20 <contract file>
```

Examples

- ERC20 Interface (erc20_interface.sol)

```
pragma lity ^1.2.4;

contract ERC20Interface {
  function totalSupply() public view returns (uint);
  function balanceOf(address tokenOwner) public view returns (uint balance);
  function allowance(address tokenOwner, address spender) public view returns (uint␣
↪remaining);
  function transfer(address to, uint tokens) public returns (bool success);
  function approve(address spender, uint tokens) public returns (bool success);
  function transferFrom(address from, address to, uint tokens) public returns (bool␣
↪success);

  event Transfer(address indexed from, address indexed to, uint tokens);
  event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

- Expect output

```
$ lityc --contract-standard ERC20 erc20_interface.sol

erc20_standard.sol:3:1: Info: ERC20Interface is compatible to ERC20.
contract ERC20Interface {
^ (Relevant source part starts here and spans across multiple lines).
```

- ERC20 Interface with wrong mutability (wrong_mutability.sol)

```
pragma lity ^1.2.4;

contract ERC20Interface {
  function totalSupply() public pure returns (uint); // mutability should be view,␣
↪not pure
```

(continues on next page)

(continued from previous page)

```
  function balanceOf(address tokenOwner) public view returns (uint balance);
  function allowance(address tokenOwner, address spender) public view returns (uint␣
↪remaining);
  function transfer(address to, uint tokens) public returns (bool success);
  function approve(address spender, uint tokens) public returns (bool success);
  function transferFrom(address from, address to, uint tokens) public returns (bool␣
↪success);

  event Transfer(address indexed from, address indexed to, uint tokens);
  event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

- Expect output

```
$ lityc --contract-standard ERC20 wrong_mutability.sol

wrong_mutability.sol:3:1: Info: Missing 'totalSupply' with type signature 'function␣
↪() view external returns (uint256)'. ERC20Interface is not compatible to ERC20.
contract ERC20Interface {
^ (Relevant source part starts here and spans across multiple lines).
```

ERC223 Contract Standard Checker

Enable contract standard checker with specific ERC

```
$ lityc --contract-standard ERC223 <contract file>
```

Examples

- ERC223 Interface (erc223_interface.sol)

```
pragma lity ^1.2.4;

contract ERC223Interface {
  function totalSupply() public view returns (uint);
  function balanceOf(address tokenOwner) public view returns (uint balance);
  function transfer(address to, uint tokens) public returns (bool success);
  function transfer(address to, uint tokens, bytes data) public returns (bool␣
↪success);
  event Transfer(address indexed from, address indexed to, uint tokens, bytes data);
}
```

- Expect output

```
$ lityc --contract-standard ERC223 erc223_interface.sol

erc223_standard.sol:3:1: Info: ERC223Interface is compatible to ERC223.
contract ERC223Interface {
^ (Relevant source part starts here and spans across multiple lines).
```

- ERC223 Token example (erc223_token.sol)

```
pragma lity ^1.2.4;

library SafeMath {
  function mul(uint a, uint b) internal pure returns (uint) {
  uint c = a * b;
  _assert(a == 0 || c / a == b);
  return c;
}

function div(uint a, uint b) internal pure returns (uint) {
  // _assert(b > 0); // Solidity automatically throws when dividing by 0
  uint c = a / b;
  // _assert(a == b * c + a % b); // There is no case in which this doesn't hold
  return c;
}

function sub(uint a, uint b) internal pure returns (uint) {
  _assert(b <= a);
  return a - b;
}

function add(uint a, uint b) internal pure returns (uint) {
  uint c = a + b;
  _assert(c >= a);
  return c;
}

function max64(uint64 a, uint64 b) internal pure returns (uint64) {
  return a >= b ? a : b;
}

function min64(uint64 a, uint64 b) internal pure returns (uint64) {
  return a < b ? a : b;
}

function max256(uint256 a, uint256 b) internal pure returns (uint256) {
  return a >= b ? a : b;
}

function min256(uint256 a, uint256 b) internal pure returns (uint256) {
  return a < b ? a : b;
}

function _assert(bool assertion) internal pure {
  if (!assertion) {
    revert();
  }
}
}

contract ERC223Interface {
  function totalSupply() public view returns (uint);
  function balanceOf(address tokenOwner) public view returns (uint balance);
  function transfer(address to, uint tokens, bytes data) public returns (bool␣
↪success);
  function transfer(address to, uint tokens) public returns (bool success);
  event Transfer(address indexed from, address indexed to, uint tokens, bytes data);
```

(continues on next page)

(continued from previous page)

```
}

contract ERC223ReceivingContract {
  function tokenFallback(address _from, uint _value, bytes _data) public;
}

contract ERC223Token is ERC223Interface {
  using SafeMath for uint;

  mapping(address => uint) balances; // List of user balances.

  function totalSupply() public view returns (uint) {
    return 2**18;
  }

  function transfer(address _to, uint _value, bytes _data) public returns (bool) {
    // Standard function transfer similar to ERC20 transfer with no _data .
    // Added due to backwards compatibility reasons .
    uint codeLength;

    assembly {
      // Retrieve the size of the code on target address, this needs assembly .
      codeLength := extcodesize(_to)
    }

    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    if(codeLength>0) {
      ERC223ReceivingContract receiver = ERC223ReceivingContract(_to);
      receiver.tokenFallback(msg.sender, _value, _data);
      return true;
    }
    emit Transfer(msg.sender, _to, _value, _data);
    return true;
  }

  function transfer(address _to, uint _value) public returns (bool) {
    uint codeLength;
    bytes memory empty;

    assembly {
      // Retrieve the size of the code on target address, this needs assembly .
      codeLength := extcodesize(_to)
    }

    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    if(codeLength>0) {
      ERC223ReceivingContract receiver = ERC223ReceivingContract(_to);
      receiver.tokenFallback(msg.sender, _value, empty);
      return true;
    }
    emit Transfer(msg.sender, _to, _value, empty);
    return true;
  }

  function balanceOf(address _owner) public view returns (uint balance) {
```

(continues on next page)

(continued from previous page)

```
    return balances[_owner];
  }
}
```

- Expect output

```
$ lityc --contract-standard ERC223 erc223_token.sol

erc223_token.sol:6:1: Info: Missing 'totalSupply' with type signature 'function ()␣
↪view external returns (uint256)'. SafeMath is not compatible to ERC223.
library SafeMath {
^ (Relevant source part starts here and spans across multiple lines).
erc223_token.sol:54:1: Info: ERC223Interface is compatible to ERC223.
contract ERC223Interface {
^ (Relevant source part starts here and spans across multiple lines).
erc223_token.sol:65:1: Info: Missing 'totalSupply' with type signature 'function ()␣
↪view external returns (uint256)'. ERC223ReceivingContract is not compatible to␣
↪ERC223.
contract ERC223ReceivingContract {
^ (Relevant source part starts here and spans across multiple lines).
erc223_token.sol:78:1: Info: ERC223Token is compatible to ERC223.
contract ERC223Token is ERC223Interface {
^ (Relevant source part starts here and spans across multiple lines).
```

ERC721 Contract Standard Checker

Enable contract standard checker with specific ERC

```
$ lityc --contract-standard ERC721 <contract file>
```

Examples

- ERC721 Interface (erc721_interface.sol)

```
pragma lity ^1.2.4;

contract ERC721Interface {
  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);
  event ApprovalForAll(address indexed _owner, address indexed _operator, bool _
↪approved);
  function balanceOf(address _owner) external view returns (uint256);
  function ownerOf(uint256 _tokenId) external view returns (address);
  function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data)␣
↪external payable;
  function safeTransferFrom(address _from, address _to, uint256 _tokenId) external␣
↪payable;
  function transferFrom(address _from, address _to, uint256 _tokenId) external␣
↪payable;
  function approve(address _approved, uint256 _tokenId) external payable;
  function setApprovalForAll(address _operator, bool _approved) external;
  function getApproved(uint256 _tokenId) external view returns (address);
```

(continues on next page)

(continued from previous page)

```
  function isApprovedForAll(address _owner, address _operator) external view returns␣
↪(bool);
  function supportsInterface(bytes4 interfaceID) external view returns (bool);
}
```

- Expect output

```
$ lityc --contract-standard ERC721 erc721_interface.sol
erc721_interface.sol:3:1: Info: ERC721Interface is compatible to ERC721.
contract ERC721Interface {
^ (Relevant source part starts here and spans across multiple lines).
```

- ERC721 Interface with wrong modification level (wrong_modification_level.sol)

```
pragma lity ^1.2.4;

contract ERC721Interface {
  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);
  event ApprovalForAll(address indexed _owner, address indexed _operator, bool _
↪approved);
  function balanceOf(address _owner) external view returns (uint256);
  function ownerOf(uint256 _tokenId) external view returns (address);
  function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data)␣
↪external payable;
  function safeTransferFrom(address _from, address _to, uint256 _tokenId) external␣
↪payable;
  function transferFrom(address _from, address _to, uint256 _tokenId) external; //␣
↪missing payable
  function approve(address _approved, uint256 _tokenId) external payable;
  function setApprovalForAll(address _operator, bool _approved) external;
  function getApproved(uint256 _tokenId) external view returns (address);
  function isApprovedForAll(address _owner, address _operator) external view returns␣
↪(bool);
  function supportsInterface(bytes4 interfaceID) external view returns (bool);
}
```

- Expect output

```
$ lityc --contract-standard ERC721 wrong_modification_level.sol

wrong_modification_level.sol:3:1: Info: Missing 'transferFrom' with type signature
↪'function (address,address,uint256) payable external'. ERC721Interface is not␣
↪compatible to ERC721.
contract ERC721Interface {
^ (Relevant source part starts here and spans across multiple lines).
```

ERC827 Contract Standard Checker

Enable contract standard checker with specific ERC

```
$ lityc --contract-standard ERC827 <contract file>
```

Examples

- ERC827 Interface (erc827_standard_no_inheritance.sol)

```
pragma lity ^1.2.4;

contract ERC827Interface {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns␣
↪(bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
  event Transfer(address indexed from, address indexed to, uint256 value);
  function transferAndCall(address _to, uint256 _value, bytes _data) public payable␣
↪returns (bool);
  function transferFromAndCall( address _from, address _to, uint256 _value, bytes _
↪data) public payable returns (bool);
  function approveAndCall(address _spender, uint256 _value, bytes _data) public␣
↪payable returns (bool);
}
```

- Expect output

```
$ lityc --contract-standard ERC827 erc827_standard_no_inheritance.sol

erc827_standard_no_inheritance.sol:3:1: Info: ERC827Interface is compatible to ERC827.
contract ERC827Interface {
^ (Relevant source part starts here and spans across multiple lines).
```

- ERC827 Interface inheritance (erc827_inheritance.sol)

```
pragma lity ^1.2.4;

contract ERC827Base {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns␣
↪(bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
  event Transfer(address indexed from, address indexed to, uint256 value);
}
contract ERC827Interface is ERC827Base {
  function transferAndCall(address _to, uint256 _value, bytes _data) public payable␣
↪returns (bool);
  function transferFromAndCall( address _from, address _to, uint256 _value, bytes _
↪data) public payable returns (bool);
  function approveAndCall(address _spender, uint256 _value, bytes _data) public␣
↪payable returns (bool);
}
```

- Expect output

```
$ lityc --contract-standard ERC827 erc827_standard.sol

erc827_standard.sol:3:1: Info: Missing 'transferAndCall' with type signature
↪'function (address,uint256,bytes memory) payable external returns (bool)'.␣
↪ERC827Base is not compatible to ERC827.
contract ERC827Base {
^ (Relevant source part starts here and spans across multiple lines).
erc827_standard.sol:13:1: Info: ERC827Interface is compatible to ERC827.
contract ERC827Interface is ERC827Base {
^ (Relevant source part starts here and spans across multiple lines).
```

ERC884 Contract Standard Checker

Enable contract standard checker with specific ERC

```
$ lityc --contract-standard ERC884 <contract file>
```

Examples

- ERC884 Interface (erc884_interface.sol)

```
pragma lity ^1.2.4;

contract C20 {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns␣
↪(bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
  event Transfer(address indexed from, address indexed to, uint256 value);
}
contract ERC884Interface is C20 {
  event VerifiedAddressAdded( address indexed addr, bytes32 hash, address indexed␣
↪sender);
  event VerifiedAddressRemoved(address indexed addr, address indexed sender);
  event VerifiedAddressUpdated( address indexed addr, bytes32 oldHash, bytes32 hash,␣
↪address indexed sender);
  event VerifiedAddressSuperseded( address indexed original, address indexed␣
↪replacement, address indexed sender);
  function addVerified(address addr, bytes32 hash) public;
  function removeVerified(address addr) public;
  function updateVerified(address addr, bytes32 hash) public;
  function cancelAndReissue(address original, address replacement) public;
  function transfer(address to, uint256 value) public returns (bool);
  function transferFrom(address from, address to, uint256 value) public returns␣
↪(bool);
  function isVerified(address addr) public view returns (bool);
  function isHolder(address addr) public view returns (bool);
  function hasHash(address addr, bytes32 hash) public view returns (bool);
  function holderCount() public view returns (uint);
```

(continues on next page)

(continued from previous page)

```
  function holderAt(uint256 index) public view returns (address);
  function isSuperseded(address addr) public view returns (bool);
  function getCurrentFor(address addr) public view returns (address);
}
```

- Expect output

```
$ lityc --contract-standard ERC884 erc884_standard.sol

erc884_standard.sol:3:1: Info: Missing 'VerifiedAddressAdded' with type signature
↪'function (address,bytes32,address)'. C20 is not compatible to ERC884.
contract C20 {
^ (Relevant source part starts here and spans across multiple lines).
erc884_standard.sol:13:1: Info: ERC884Interface is compatible to ERC884.
contract ERC884Interface is C20 {
^ (Relevant source part starts here and spans across multiple lines).
```

- ERC884 Interface with missing function (missing_function.sol)

```
pragma lity ^1.2.4;

contract ERC884Interface {
  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);
  event ApprovalForAll(address indexed _owner, address indexed _operator, bool _
↪approved);
  function balanceOf(address _owner) external view returns (uint256);
  function ownerOf(uint256 _tokenId) external view returns (address);
  function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data)␣
↪external payable;
  function safeTransferFrom(address _from, address _to, uint256 _tokenId) external␣
↪payable;
  function transferFrom(address _from, address _to, uint256 _tokenId) external; //␣
↪missing payable
  function approve(address _approved, uint256 _tokenId) external payable;
  function setApprovalForAll(address _operator, bool _approved) external;
  function getApproved(uint256 _tokenId) external view returns (address);
  function isApprovedForAll(address _owner, address _operator) external view returns␣
↪(bool);
  function supportsInterface(bytes4 interfaceID) external view returns (bool);
}
```

- Expect output

```
$ lityc --contract-standard ERC884 missing_function.sol

missing_function.sol:3:1: Info: Missing 'VerifiedAddressAdded' with type signature
↪'function (address,bytes32,address)'. ERC884Interface is not compatible to ERC884.
contract ERC884Interface {
^ (Relevant source part starts here and spans across multiple lines).
```

2.5 FAQ

2.6 Developer's Guide

2.6.1 Build Lity from source

Download Source Code

```
git clone https://github.com/CyberMiles/lity.git
```

Install Dependencies

In your lity directory:

```
./scripts/install_deps.sh
```

Build with CMake

In your lity directory:

```
mkdir build
cd build
cmake ..
make
./lityc/lityc --version
```

Run Tests

In your lity directory:

```
./build/test/soltest -- --testpath ./test --no-ipc --show_progress
./test/cmdlineTests.sh
```

2.7 How to Contribute

Help is always appreciated!

This document briefly explains the process to report an issue or submit a pull request.

See the contribution guidelines in the Solidity documentation for more information.

2.7.1 Report an Issue

Please provide the following information as much as possible.

- The version (commit-ish) your using.
- Your platform, environment setup, etc.

- Steps to reproduce the issue.
- Your expected result of the issue.
- Current result of the issue.

2.7.2 Create a Pull Request

- Fork from the *lity* branch.
- Avoid to create merge commits when you update from *lity* by using `git rebase` or `git pull --rebase` (instead of `git merge`).
- Add test cases for your pull request if you're proposing new features or major bug fixes.
- Build and test locally before submit your pull request.
- Respect the coding style for this project.

2.8 News

2.9 Events