Lity Documentation

CyberMiles

Sep 04, 2018

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 accelerate executing time. You can now encrypt large-sized data in the blink of an eye without running out of gas anymore.

1.1 How Lity Enhances Security

With ERC Checker and Overflow Protection.

1.2 How Lity Improves Performance

With Ethereum Native Interface (ENI).

See libENI on Github and the libENI documentation.

CHAPTER 2

Lity by Example

Here we will provide a contract example and explain a little bit.

```
pragma solidity ^0.4.23;

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

CHAPTER 3

Contents

3.1 Getting Started with Lity

See the documentation for how to download and install Lity.

3.1.1 Use Lityc to Secure Your Contract

Some instructions to use ERC checker and overflow protection.

3.1.2 Use ENI in Your Contract

ENI Tutorial

3.2 How to Download Lity

3.2.1 Build from source

```
git clone https://github.com/CyberMiles/lity.git
cd lity
mkdir build
cd build
sudo apt-get install cmake libblkid-dev e2fslibs-dev libboost-all-dev libaudit-dev
cmake ..
make
./lityc/lityc --help
```

3.3 What's Special About Lity

3.3.1 Better Security

Make your contract more secure with

- *ERC Checker*, and
- *Overflow Protection*.

ERC Checker

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.

3.3.2 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.

3.3.3 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 tutorial for writing Lity rules.

3.4 Documentation

3.4.1 Beginner's Guide

Beginner's Guide

3.4.2 Developer's Guide

Developer's Guide

3.4.3 Docs

Rule Engine

- Lity Rule by Examples
- Lity rule engine specs

Types

- types-fixed-point-numbers

Security & Safe

ERC Contract Standard Checker

- ERC20 Checker
- ERC223 Checker
- ERC721 Checker
- ERC827 Checker
- ERC884 Checker

Overflow Protection

- safeuint - a new type with native SafeMath support
- Bec token with overflow protection

Validator-Only Contract

- Validator-Only Contract

Analysis Tool

- Oyente Integration

Performant & Flexible

Schedule Transaction

- Schedule Transaction

Ethereum Native Interface (ENI) Tutorial

- Tutorial

ENI Examples

- Reverse String
- Verify Dogecoin Block on Travis
- RSA encryption and decryption

3.4.4 FAQ

FAQ

3.5 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.

3.5.1 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.

3.5.2 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.

3.6 News

3.7 Events