
devops199 commented 22 hours ago • edited
I accidentally killed it.
https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4
¯_(ツ)_/¯

Famous bugs in ethereum smartcontracts

- DAO, default function reentrancy → 50M$ stolen
- parity multisig, internal initialization function was public → 30M$ stolen
- parity multisig, uninitialized library → 150M$ locked
- smartbillions lottery, RTFM → 120K$ bounty
- hackergold, =+ instead +=

Improve technical debt ?

Learn about

- secure systems and cryptography, now zk-snarks?
- game theory + incentivisation systems + financial markets
- architecture and scalability
- real-world applications, web, mobile, web3
- specific smartcontract security
- state of the network
- lot of changes

Improve solidity ?

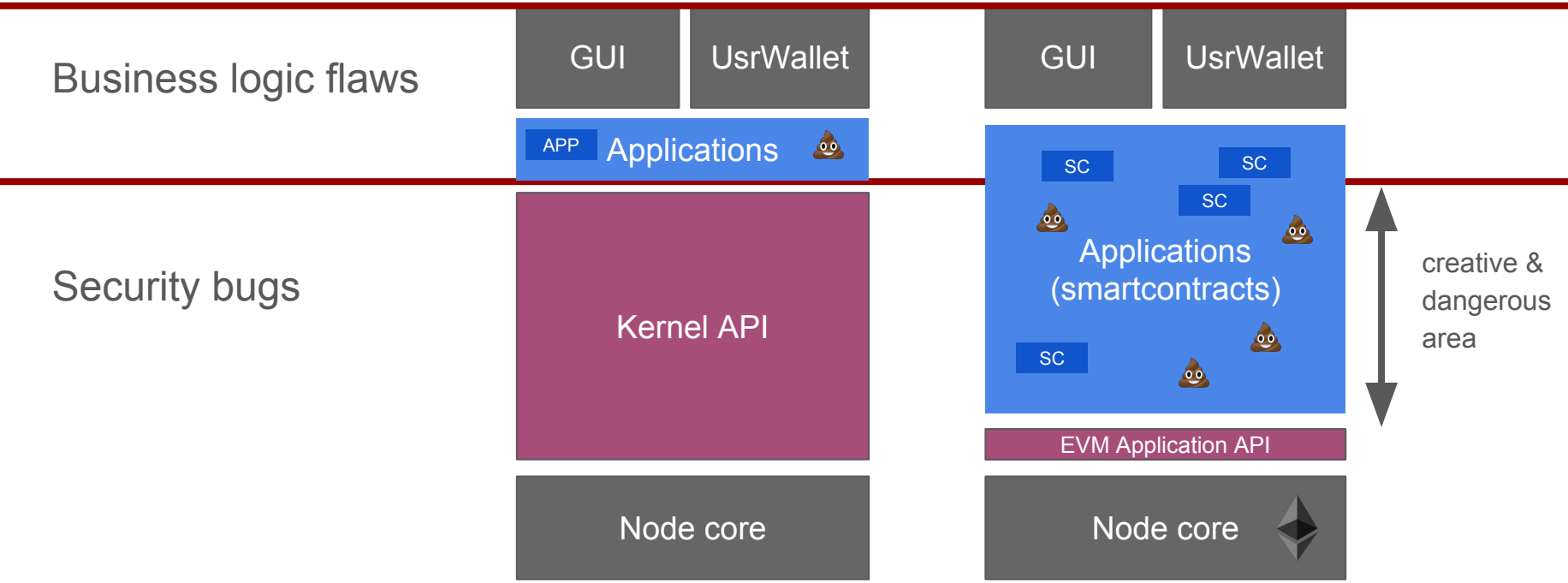
```
for (var i=0;i<1000;i++) …

function u() returns (bool) {
        return a;
        bool a = true;
}

uint a=0; a--;
```

diamond OOP pattern is not a very safe practice...

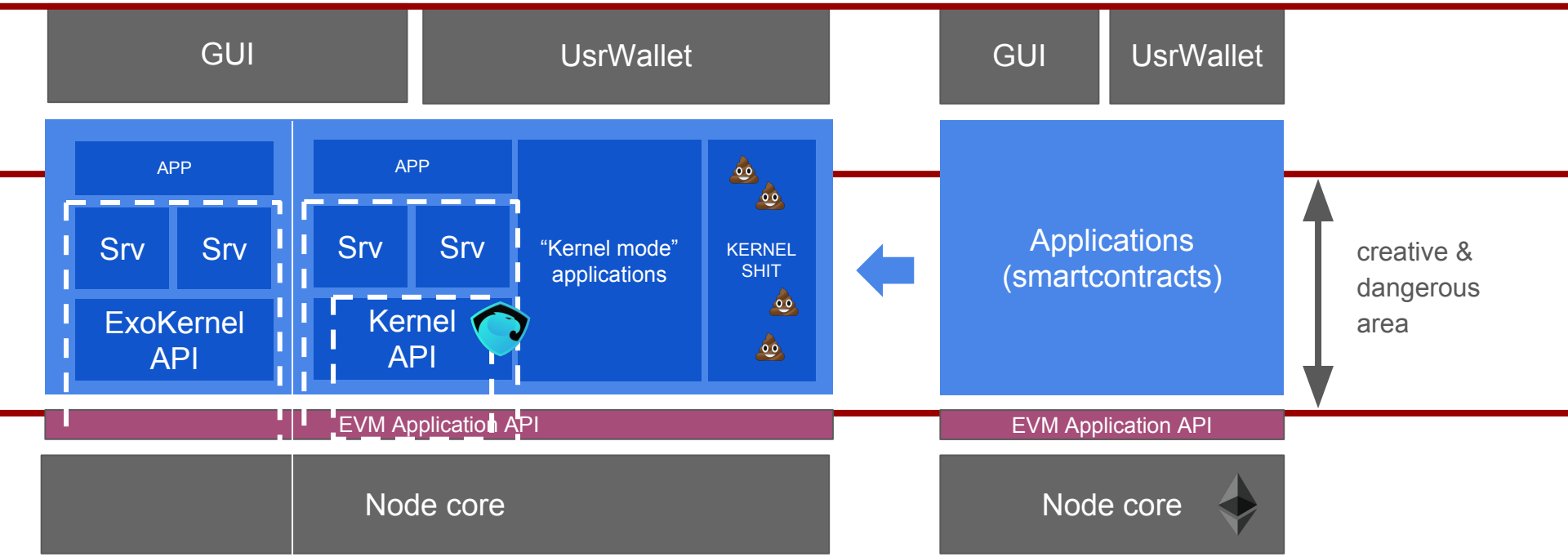
Improve community ?

- Coordinate security advisories
  - A bug found in the EVM compiler?
  - A bug found in a commonly used library?
- No consensus on security audits
- Seems that nobody cares about bug bounties
- Sometimes smartcontracts does not have the proper documentation to use them

Improve security arch?

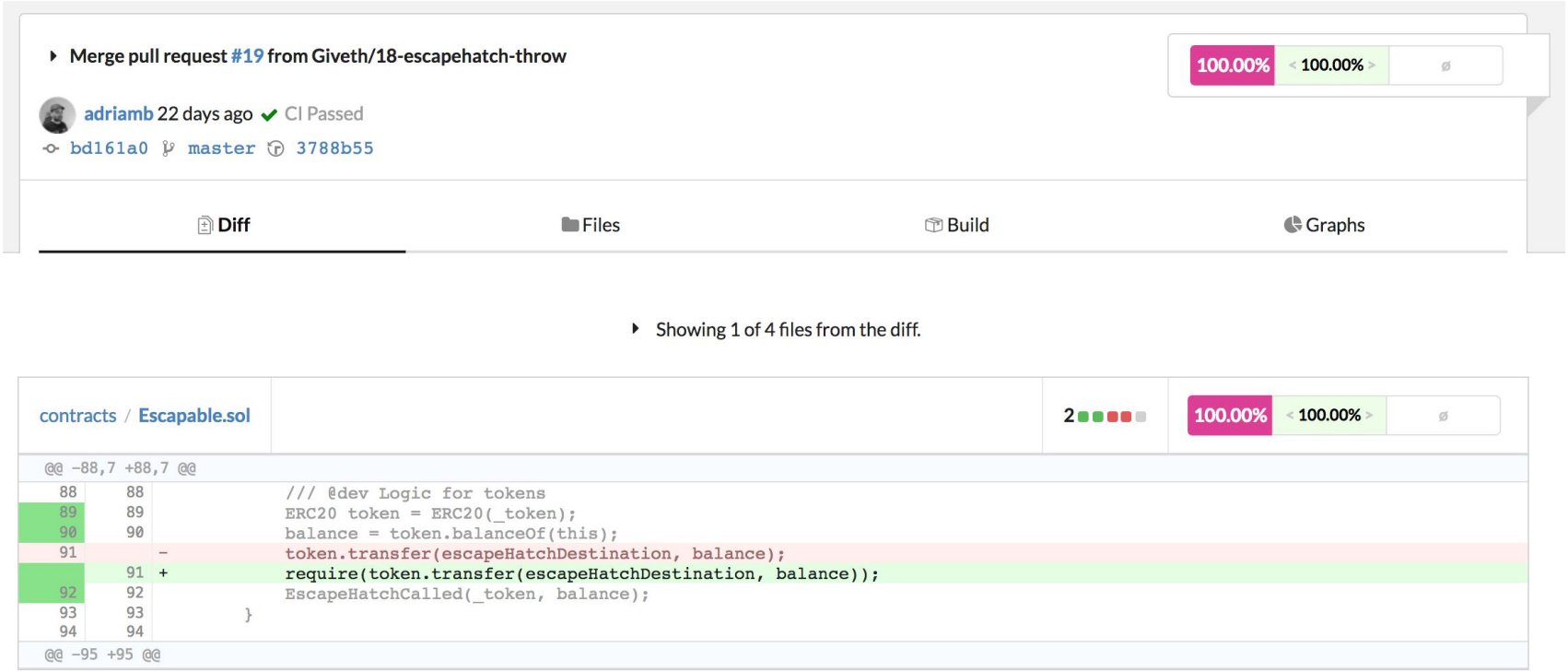


Kernels welcomed



sharding-driven kernel isolation & kernel-rings isolation provided by EVM via privileged OPCODES or EVMonEVM?

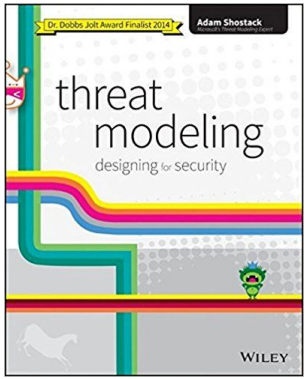
Code coverage & linters

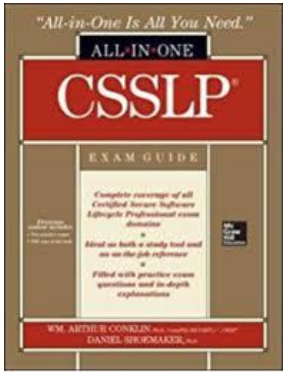


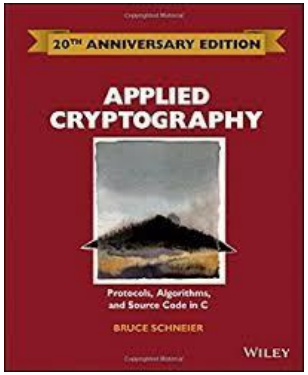
see https://github.com/Giveth/common-contract-deps

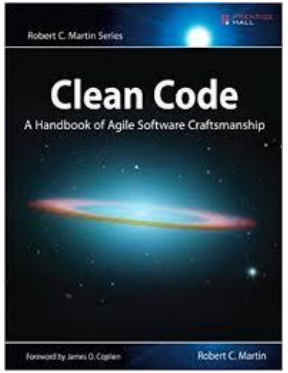
Nice community and lot of information

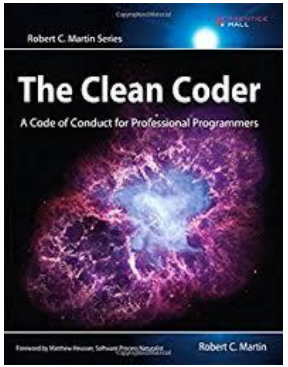
- a ton of code reviews to learn from
- http://u.solidity.cc/ Underhanded Solidity Coding Contest
- https://consensys.github.io/smart-contract-best-practices
- very good documentation https://solidity.readthedocs.io











More security awareness welcomed

```
pragma solidity 0.4.14;
contract SimpleMultiSig {

  uint public nonce;               // (only) mutable state
  uint public threshold;           // immutable state
  mapping (address => bool) isOwner; // immutable state
  address[] public ownersArr;        // immutable state

  function SimpleMultiSig(uint threshold_, address[] owners_) {
    if (owners_.length > 10 || threshold_ > owners_.length || threshold_ == 0) {throw;}

    for (uint i=0; i<owners_.length; i++) {
      isOwner[owners_[i]] = true;
    }
    ownersArr = owners_;
    threshold = threshold_;
  }

  // Note that address recovered from signatures must be strictly increasing
  function execute(uint8[] sigV, bytes32[] sigR, bytes32[] sigS, address destination, uint value, bytes data) {
    if (sigR.length != threshold) {throw;}
    if (sigR.length != sigS.length || sigR.length != sigV.length) {throw;}

    // Follows ERC191 signature scheme: https://github.com/ethereum/EIPs/issues/191
    bytes32 txHash = sha3(byte(0x19), byte(0), this, destination, value, data, nonce);

    address lastAdd = address(0); // cannot have address(0) as an owner
    for (uint i = 0; i < threshold; i++) {
        address recovered = ecrecover(txHash, sigV[i], sigR[i], sigS[i]);
        if (recovered <= lastAdd || !isOwner[recovered]) throw;
        lastAdd = recovered;
    }

    // If we make it here all signatures are accounted for
    nonce = nonce + 1;
    if (!destination.call.value(value)(data)) {throw;}
  }

  function () payable {}
}
```

formally verificable multisig, https://medium.com/@ChrisLundkvist/exploring-simpler-ethereum-multisig-contracts-b71020c19037

Correctness proofs : from idea to execution

```
pragma solidity ^0.4.15;

contract SmartContract {
    function splitEqually() {
        var (sale,remain) = split(msg.value);
        assert(sale+remain==msg.value);
    }
    ...
}
```

```
6060604052341562000010576000
80fd5b6040516200191438038062
0019148339810160405280805182
0191906020018051906020019091
9050506000825182603282118062
00004e57508181115b806200005a
5750600081145b80620000665750
600082145b156200007157600080
fd5b600092505b84518310156200
01a857600260008658151811015
156200009357fe5b906020019060
2002015173ffffffffffffffffff
ffffffffffffffffffffffff1673ff
ffffffffffffffffffffffffffffff
ffffffffff16815260200190815260
6020016000206000905490610100
0a900460ff16806200011f575060
0085848151811015156200000fd57
fe5b9060200190602002015173ff
ffffffffffffffffffffffffffff
ffffffffff16145b156...
```

EVM

Correctness proofs : formal methods

```
function infactorial(uint val, uint N) returns (bool found);
```

$$infact : v, N \mapsto \exists \alpha : 0 < \alpha \leq N \mid v = \alpha!$$

Correctness proofs : formal methods

preconditions, invariants, postconditions

```
function infactorial(uint val, uint N) returns (bool found) {
    /// pre: 0 < N < 40
   bool found = false
   uint v = 1;
   for (uint i=1;i<=N;i++) {
       v *= i;
       if (val == v) found = true;
       /// inv: v = i!
       /// inv: found ≡ ∃α:0<α≤i|val=α!
   }
    /// post: found ≡ ∃α:0<α≤N|val=α!
}
```

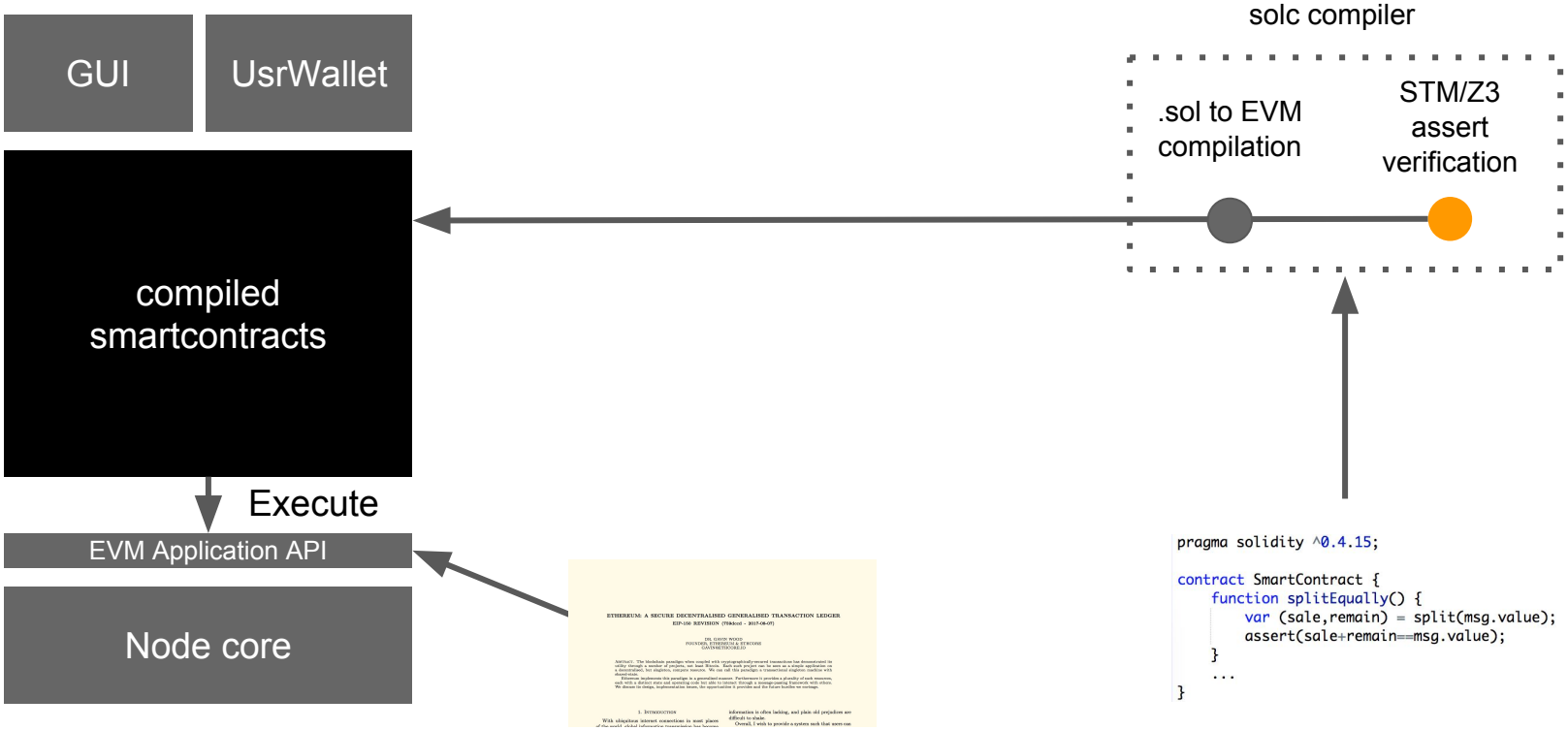
$$infact : v, N \mapsto \exists \alpha : 0 < \alpha \leq N \mid v = \alpha!$$

Correctness proofs : haskell modelling?

When a CDP has no risk problems (except that its ilk's ceiling may be exceeded), its owner can use `free` to reclaim some amount of collateral, as long as this would not take the CDP below its liquidation ratio.

```
free id_urn wad_gem = do
  Fail if sender is not the CDP owner
  id_lad ← use sender
  owns id_urn id_lad
  Record a collateral token balance decrease
  decrease (urns ∘ ix id_urn ∘ ink) wad_gem
  Roll back on any risk problem except ilk ceiling excess
  want (feel id_urn) (oneOf [Pride, Anger])
  Release custody of collateral
  id_ilk ← look (urns ∘ ix id_urn ∘ ilk)
  id_tag ← look (ilks ∘ ix id_ilk ∘ gem)
  transfer (Gem id_tag) wad_gem Jar id_lad
```

Correctness proofs : SMT/Z3 theorem prover



https://github.com/kframework/evm-semantics

https://www.ideals.illinois.edu/bitstream/handle/2142/97207/hildenbrandt-saxena-zhu-rodrigues-guth-daian-rosu-2017-tr_0818.pdf

Correctness proofs : SMT/Z3 theorem prover

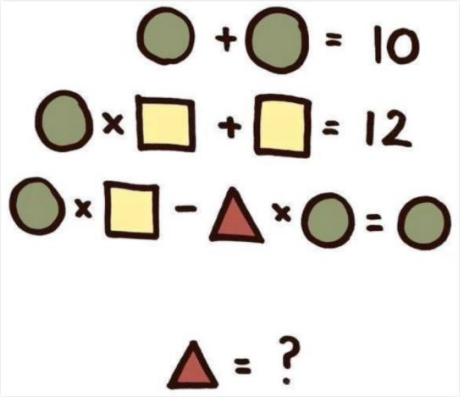
```
pragma experimental SMTChecker;
contract SimpleExample {
  function add() pure public returns (uint) {
    for (uint i = 10; i >= 0; i--) {
      // ...
    }
  }
}
```

```
:4:23: Warning: For loop condition is always true.
  for (uint i = 10; i >= 0; i--) {
                    ^----^
```

Assumes "require" expressions and tries to prove:

- assertions
- no overflows / underflows
- no division by zero
- no constant conditions / unreachable code
- ...

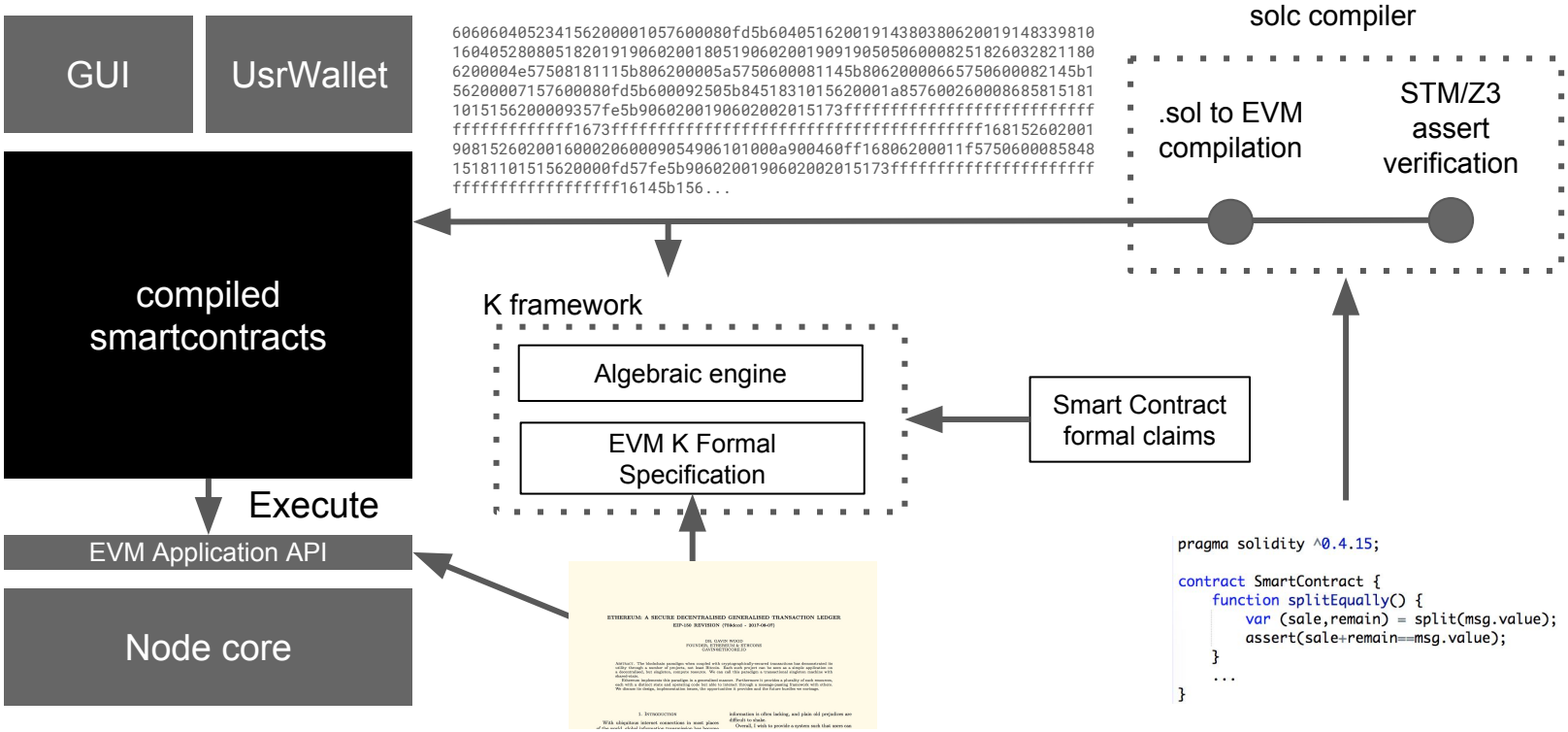
Correctness proofs : SMT/Z3 theorem prover




```
pragma experimental SMTChecker;
contract SimpleExample {
  function f(int16 circle, int16 square,
             int16 triangle) pure public {
    require(circle + circle == 10);
    require(circle * square + square == 12);
    require(circle * square -
            triangle * circle == circle);
    assert(false);
  }
}
```

```
:7:5: Warning: Assertion violation happens here for:
circle = 5
square = 2
triangle = 1

  assert(false);
  ^-----------^
```

Correctness proofs : KEVM



https://github.com/kframework/evm-semantics

https://www.ideals.illinois.edu/bitstream/handle/2142/97207/hildenbrandt-saxena-zhu-rodrigues-guth-daian-rosu-2017-tr_0818.pdf

Tu has retuitat

Programming Wisdom @CodeWisdom · 26 oct.

"Every great developer you know got there by solving problems they were unqualified to solve until they actually did it." - Patrick McKenzie

Tradueix del anglès

11 1,5m 3,1m

adria@codecontext.io / @codecontext