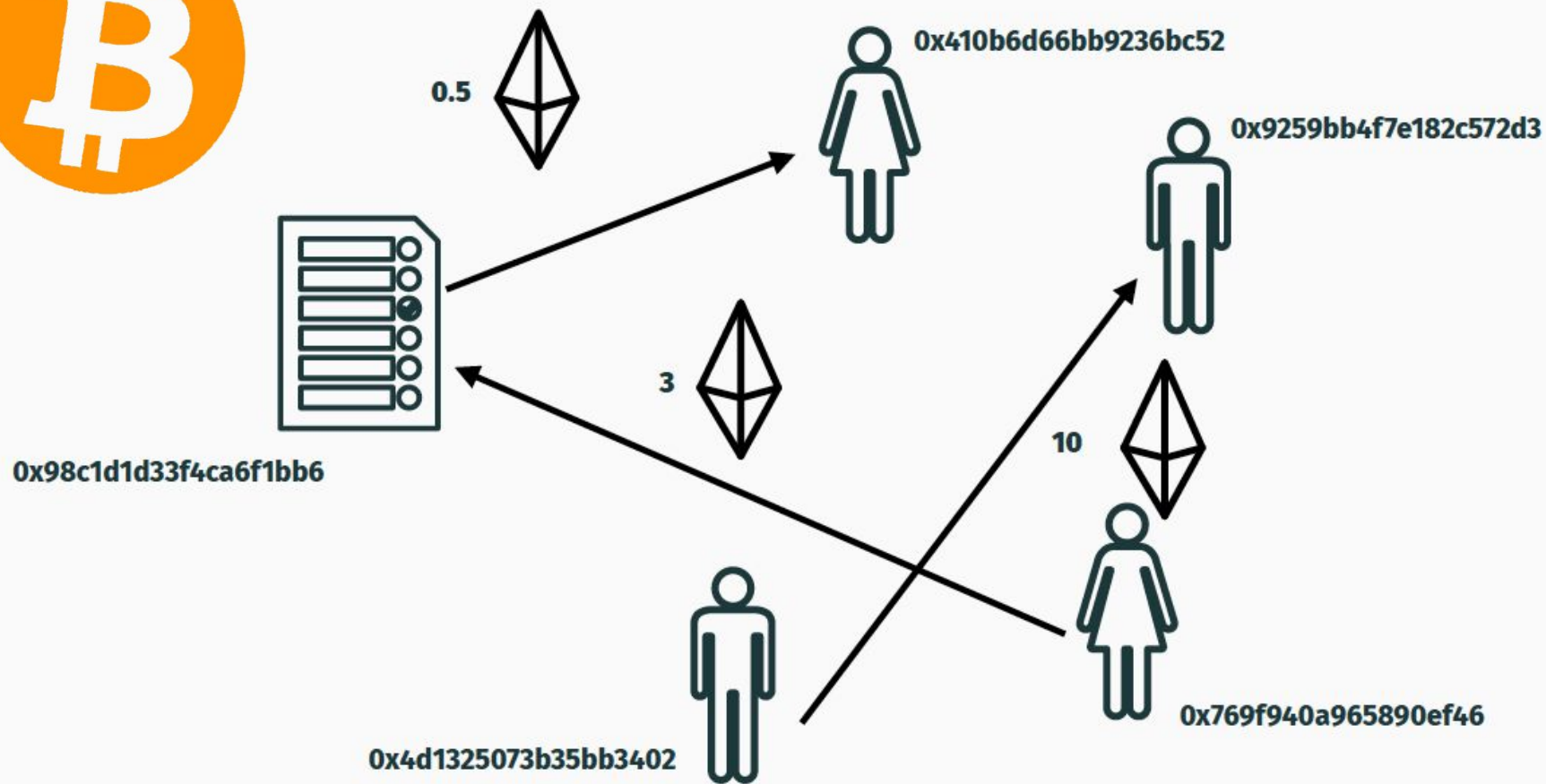


# EthBMC: A Bounded Model Checker for Smart Contracts

Joel Frank, Cornelius Aschermann, Thorsten Holz  
Usenix Sec'20



If event X happens



0x410b6d66bb9236bc52



0x98c1d1d33f4ca6f1bb6



0x9259bb4f7e182c572d3

10

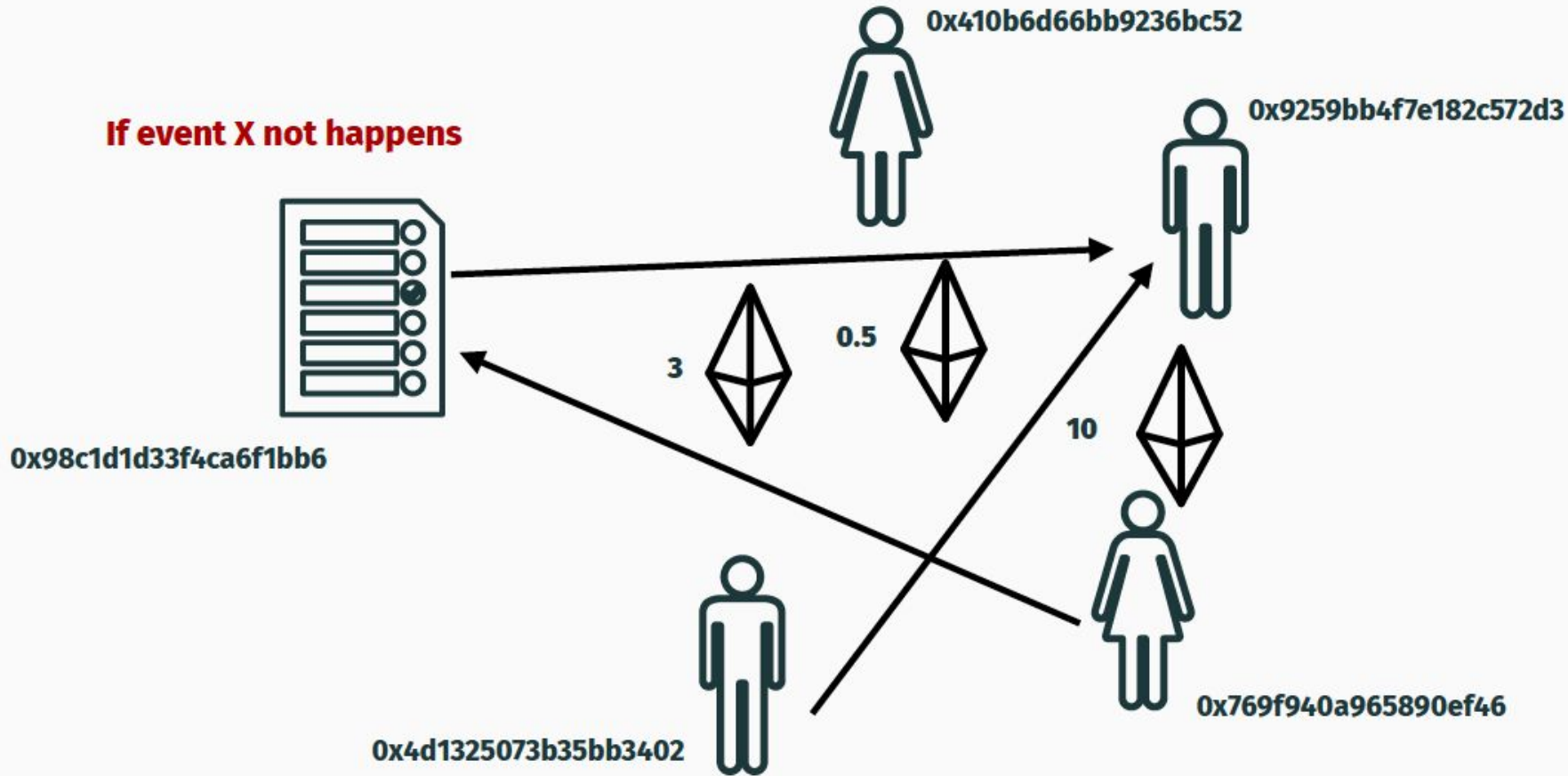


0x4d1325073b35bb3402



0x769f940a965890ef46

**If event X not happens**



A problem has been detected and Windows has been shut down to prevent damage to your computer.

## **A \$50 Million Hack Just Showed That the DAO Was All Too Human**

**\$30 Million: Ether Reported Stolen Due  
to Parity Wallet Breach**

your computer, press F8 to select Advanced Startup Options, and then select

Techn

\*\*\* S

\*\*\* k

**Someone 'Accidentally' Locked Away  
\$150M Worth of Other People's Ethereum  
Funds**

... ..

# Contribution

1. provide a survey of the current state-of-the-art analyzers for the Ethereum network, finding all of them to lack precise reasoning over EVM internals
2. present the design and implementation of ETHBMC, a bounded model checker which handles the identified issues by more precisely reasoning about the internals of EVM
  - a. analyzing symbolic memcopy-style operations
  - b. inter-contract communication
  - c. precisely reasoning about cryptographic hash functions
3. implemented a prototype of ETHBMC in 13,000 lines of Rust code and demonstrate its capabilities in several experiments.

# Common obstacles & A Toy Example


1. Keccak256 function
2. Memcopy-like instructions
3. Inter-Contract Communication
4. Toy example: Parity Wallet Bug

# Keccak256 function

```
1  function solve(uint256 input) {  
2      if (keccak256(input) == 0x315dd8 ...)   
3          selfdestruct(msg.sender);  
4      }  
5  }
```

1. Keccak function is invoked via keccak256 key word. Solidity-based smart contracts make intensive use of this instruction when implementing the mapping data type, e.g., hash table.
2. An example: the instruction is used to calculate a memory location
  - a. The attacker could generate a valid hash collision function to overwritten the value

```
1  mapping(uint => address) map;  
2  function createUser(address addr, uint id) public {  
3      map[id] = addr;  
4  }  
5  function destruct(uint id) public {  
6      if (map[id] == msg.sender) {  
7          selfdestruct(msg.sender);  
8      }  
9  }
```



$keccak256(k \parallel \hat{p})$



# Memcpy-like Instructions

1. EVM cannot access calldata directly, it can only operate on data residing within *execution memory*, i.e., the input data gets copied
2. E.g., EVM utilizes the CALLDATACOPY instruction to copy the entire input to execution memory. String vs. uint256 (can be contrasted)

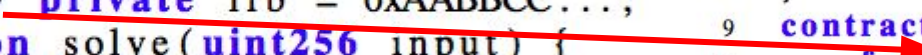
```
1  function solve(string input) {  
2      if (input[0] == "A" && input[1] == "B"){  
3          selfdestruct(msg.sender);  
4      }  
5  }
```

Listing 3: Memcopy-like operation to access input.

# Inter-Contract Communication

1. Ethereum is a decentralized system, offering the ability for multiple contracts to interact with each another. It will also increase system complexity and might lead to unforeseen (security) consequences.
2. most contracts are not deployed by humans, but rather *are created by other contracts*, making these contracts part of *intra-contract* interactions

```
1  contract Target {
2      Library private lib = 0xAABBCC...;
3      function solve(uint256 input) {
4          if (lib.r(input) == 123) {
5              selfdestruct(msg.sender);
6          }
7      }
8  }
9  contract Library {
10     function r(uint256 input) returns (ui
11         return input;
12     }
13 }
```



```

1  contract WalletLibrary {
2      address[256] owners;
3      mapping(bytes -> uint256) approvals;
4      function confirm(bytes32 _op) internal bool {
5          /* logic for confirmation */
6      }
7      function initWallet(address[] _owners) {
8          /* initialize the wallet owners */
9      }
10     function pay(address to, uint amount) {
11         if (confirm(keccak256(msg.data)))
12             to.transfer(amount);
13     }
14 }
15 contract Wallet {
16     address library = 0xAABB...;
17     // constructor
18     function Wallet(address[] _owners) {
19         library.delegatecall("initWallet", _owners)
20     }
21     function() payable {
22         library.delegatecall(msg.data);
23     }

```

Mapping

Keccak256 function

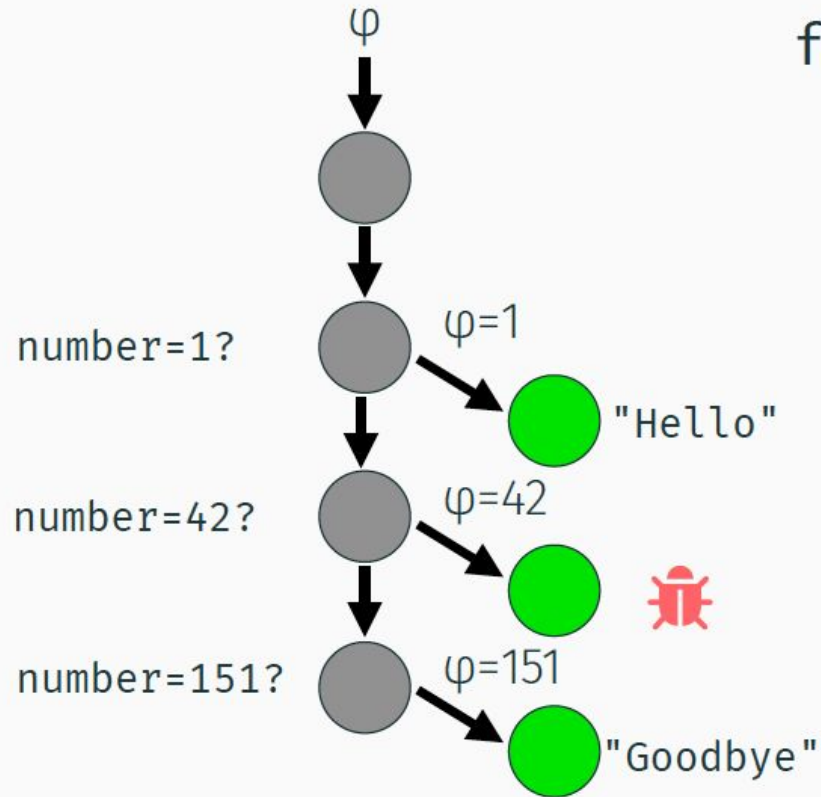
If the attacker can redirect the flow to her address, she can steal the money



Listing 5: A simplified source code from the Parity wallet.

# Comparison with State-of-the-art works

| Tool           | Inter-Contract | Memory | Keccak | Validation |
|----------------|----------------|--------|--------|------------|
| Manticore [39] | ◐              | ◐      | ◐      | ○          |
| Mythril [41]   | ◐              | ◐      | ◐      | ○          |
| MAIAN [46]     | ○              | ◐      | ◐      | ●          |
| Oyente [36]    | ○              | ◐      | ○      | ○          |
| teEther [33]   | ○              | ◐      | ◐      | ●          |
| Vandal [4]     | ○              | ◐      | ○      | ○          |
| MadMax [23]    | ○              | ◐      | ○      | ○          |
| Securify [62]  | ○              | ◐      | ◐      | ○          |
| ETHBMC         | ●              | ●      | ●      | ●          |

● Correctly implemented    ◐ Partially implemented  
○ Incorrectly implemented or missing



```
function(number):  
    if (number = 1) {  
        say "Hello"  
    }  
  
    if (number = 42) {  
          
    }  
  
    if (number = 151) {   
        say "Goodbye"  
    }
```

Satisfiability Modulo Theory (SMT) solver to check if the program path is feasible

# Attacker Model

1. ETHBMC provides a symbolic, multi-account capable representation of the Ethereum ecosystem which can be used to check arbitrary models
2. Three attack vectors:
  - a. an attacker who wants to extract Ether from the analyzed contract.
  - b. an attacker who wants to redirect the control flow of the analyzed contract to her own account
  - c. an attacker who wants to selfdestruct the analyzed contract
3. The attacker is able to participate in the Ethereum protocol, obtain a live view of the network and the blockchain, including storage and bytecode level access to contracts

# Tackling Three Obstacles

1. Encoding scheme: keccak is a binding function, i.e., when the same input is supplied to the function, it will produce the same output.
2. Fully symbolic memory model. We represent the memory as a graph representation, connecting different memory regions when we copy from one to the other (for memory copy instruction)
3. Support symbolic copy instructions, to correctly model the input memory to the call operation (for inter-contract analysis)

Validation as an additional potential obstacle; i.e., are any overapproximations correctly validated afterwards

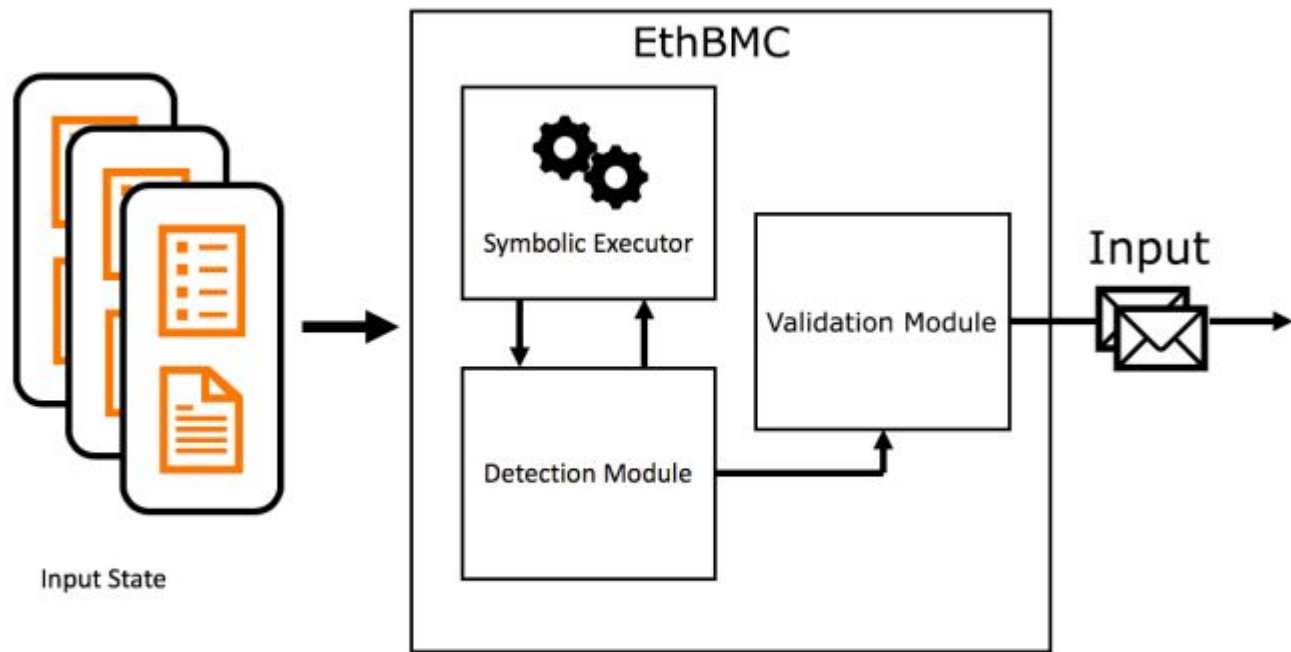


Figure 2: High-level overview of ETHBMC and its inner workings.



# Sub-Modules

1. Symbolic Executor: utilizes its symbolic execution engine to explore the available state space a program can reach, and translate the necessary conditions (or constraints) needed to reach this state into first-order logic
2. Detection Module: encodes the attacker's goal using additional constraints and utilizes the SMT solver to solve the constraint system
3. Validation Module: generates valid transactions for every state which has a feasible attack path. utilize the SMT solver to generate the transaction data needed to trigger the vulnerability

As of December 2018:

- We analysed 2.2 million smart contracts
- 4,301 Contract accounts vulnerable

Analysis:

- Successfully analysed around 90%
- We only spent 30 min / contract
- Remaining 10% are the most complex
- Also the contracts with most value

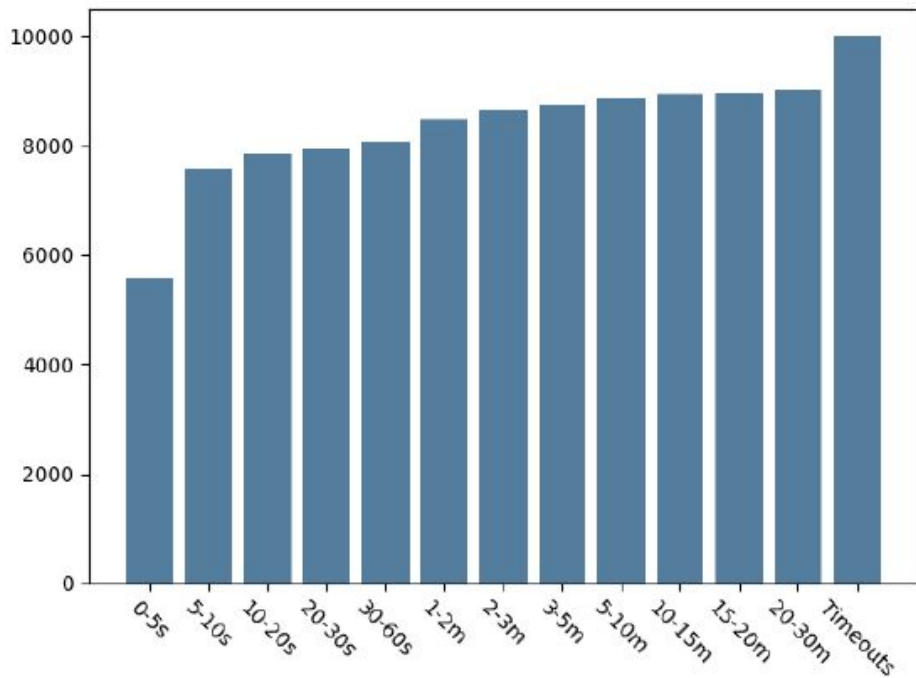


Table 3: Large-scale analysis results displaying the amount of contracts found (with the amount of unique exploits generated in brackets)

| Analyzer | Steal Ether  |         | Hijack    |       | Suicidal     |         | Total |         |
|----------|--------------|---------|-----------|-------|--------------|---------|-------|---------|
| ETHBMC   | <b>1,681</b> | (1,893) | <b>51</b> | (54)  | 1,431        | (1,474) | 2,856 | (3,367) |
| teEther  | 1,509        | (1,541) | 8         |       | -            |         | 1,509 | (1,541) |
| ETHBMC   | 1,693        | (1,964) | 51        | (54)  | <b>1,439</b> | (1,482) | 2,921 | (3,448) |
| MAIAN    | -            |         | -         |       | 1,423        |         | 1,423 |         |
| ETHBMC   | 2,708        | (3,916) | 97        | (123) | 1,924        | (1,989) | 4,301 | (5,905) |

Table 4: Ablation Study of ETHBMC

| Features        | Steal Ether |          | Hijack |       | Suicidal |          | Total |          |
|-----------------|-------------|----------|--------|-------|----------|----------|-------|----------|
| teEther         | 1,509       |          | 8      |       | -        |          | -     |          |
| Baseline ETHBMC | 1,543       |          | 50     |       | 1,403    |          | 2,709 |          |
| + Memory        | 1,557       | (+0.91%) | 51     | (+2%) | 1,409    | (+0.43%) | 2,725 | (+0.6%)  |
| + Keccak        | 1,628       | (+4.56%) | 51     |       | 1,425    | (+1.13%) | 2,803 | (+2.86%) |
| + Calls         | 1,681       | (+3.36%) | 51     |       | 1,431    | (+0.42%) | 2,856 | (+1.89%) |

# Conclusion

1. More precise analysis of smart contracts is possible
2. Scales to the majority of contracts
3. Most complex contracts still need work