# SmartMark: Software Watermarking Scheme for Smart Contracts

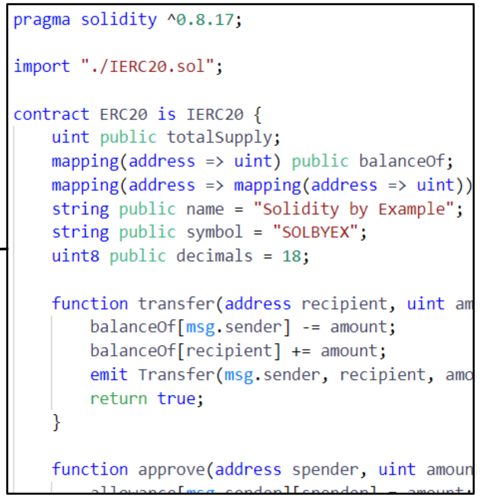
Taeyoung Kim, Yunhee Jang, Chanjong Lee, Hyungjoon Koo, Hyoungshick Kim

Sungkyunkwan University, Korea

# Smart Contract

- A self-executing program on a blockchain that ensures reliable transactions
- Its compiled bytecode is <u>publicly available</u>

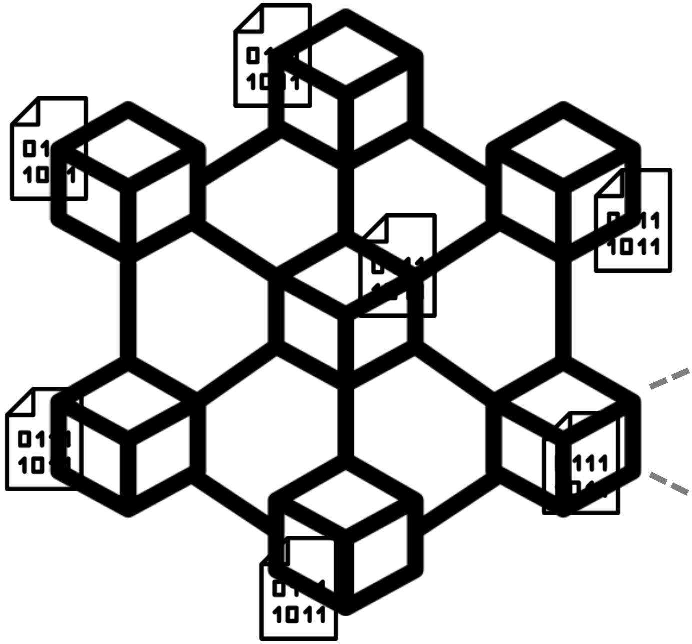
Smart contract source code (Solidity)



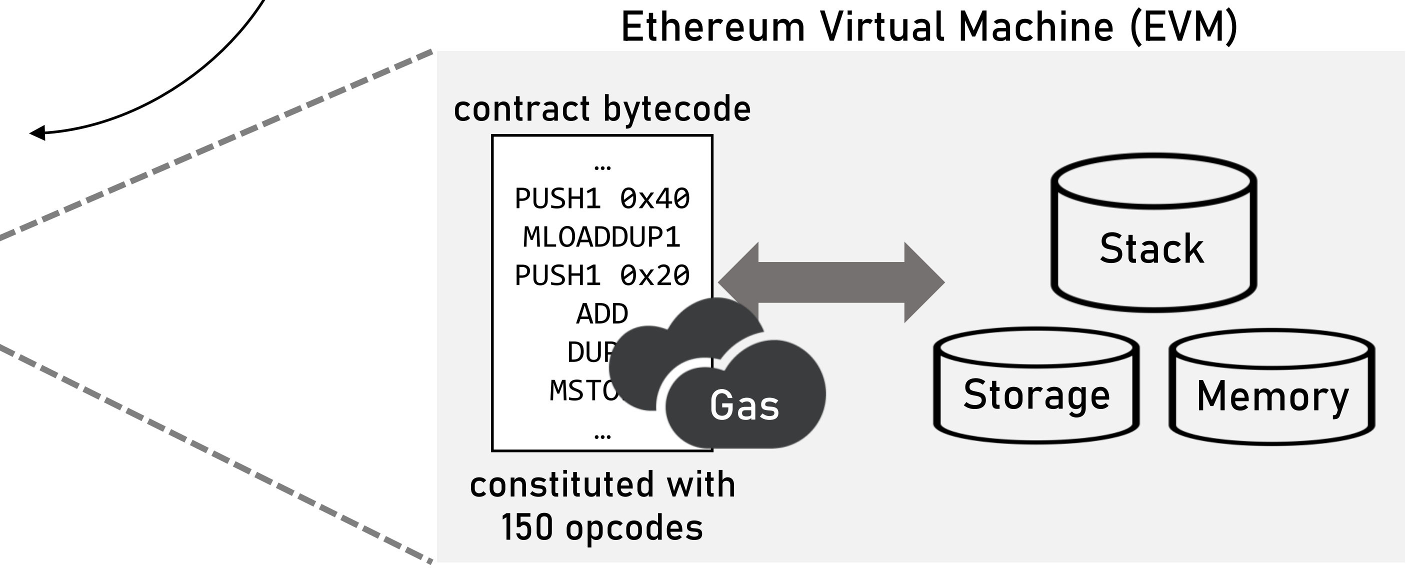
```
pragma solidity ^0.8.17;

import "./IERC20.sol";

contract ERC20 is IERC20 {
    uint public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint))
    string public name = "Solidity by Example";
    string public symbol = "SOLBYEX";
    uint8 public decimals = 18;

    function transfer(address recipient, uint am
        balanceOf[msg.sender] -= amount;
        balanceOf[recipient] += amount;
        emit Transfer(msg.sender, recipient, amo
        return true;
    }

    function approve(address spender, uint amoun
        allowance[msg.sender][spender] = amount;
```

Compilation & Deployment

Blockchain network



Ethereum Virtual Machine (EVM)



contract bytecode

```
…
PUSH1 0x40
MLOADDUP1
PUSH1 0x20
ADD
DUP
MSTO
…
```

constituted with 150 opcodes

# Smart Contract Clones in the Wild

```
DiviesInterface constant private Divies = DiviesInterface(0xc7…);
PlayerBookInterface constant private PlayerBook =
      PlayerBookInterface(…);
…
string constant public name = "FoMo3D Long Official";
```

```
      …
}
…
```

original FoMo3D

```
[-]DiviesInterface constant private Divies = DiviesInterface(0xc7…);
   PlayerBookInterface constant private PlayerBook =
        PlayerBookInterface(…);
   …
[-]string constant public name = "FoMo3D Long Official";
[+]string constant public name = “Peach Will";
```

```
[+]     uint256 _com = (_pot / 20);
        …
   }
   …
```

clone FoMo3D

To protect smart contracts from software piracy,
we propose a new software watermarking scheme for smart contracts

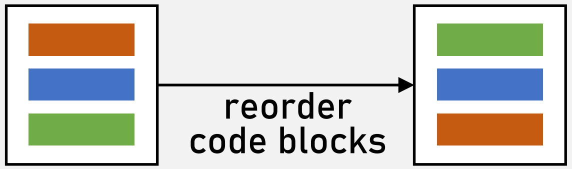
- ✓ Over 96% of 10M contracts have duplicates
- ✓ 73 DApps are plagiarized from 41 original DApps, incurring substantial financial losses[1]

However, there is no technical means to claim the smart contract originality on demand

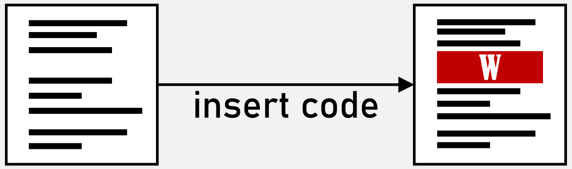
[1] N. He et al., “Characterizing Code Clones in the Ethereum Smart Contract Ecosystem,” FC, 2020

# Existing Software Watermarking Approaches

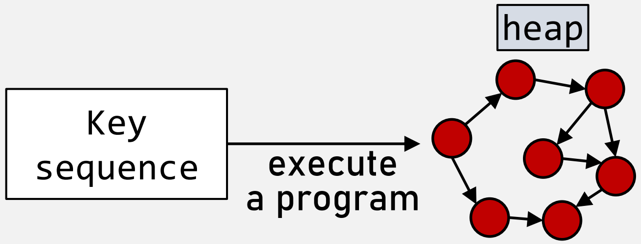
- Reordering code at the level of a block or a function[2]



- Inserting (obfuscated) dummy code[3] or (ROP) instructions[4] that make up a watermark



- Using dynamically allocated memory (*e.g.,* dynamic graph watermark)[5]



[2] H. Kang et al., “SoftMark: Software Watermarking via a Binary Function Relocation,” ACSAC, 2021
[3] A. Monden et al., “A Practical Method for Watermarking Java Programs,” COMPSAC, 2000
[4] H. Ma et al., “Software Watermarking Using Return-oriented Programming,” CCS, 2015
[5] C. Collberg et al., “Software Watermarking: Models and Dynamic Embeddings,” POPL, 1999

# Challenges in Smart Contract Watermarking

- Smart contracts have a **code size restriction** (24KB, EIP-170)
  - A smart contract might not have enough code to be reordered
  - It is hard to obfuscate dummy code against static analyses

- Running a smart contract incurs **execution costs** (gas)
  - Inserting additional watermark instructions would make contracts non-economical

- Smart contracts are executed on Ethereum Virtual Machine (EVM)
  - EVM does not support heap allocation, disabling a dynamic watermark construction

# Our Design Choices

① Use existing smart contract code
**not inserting any watermark code**

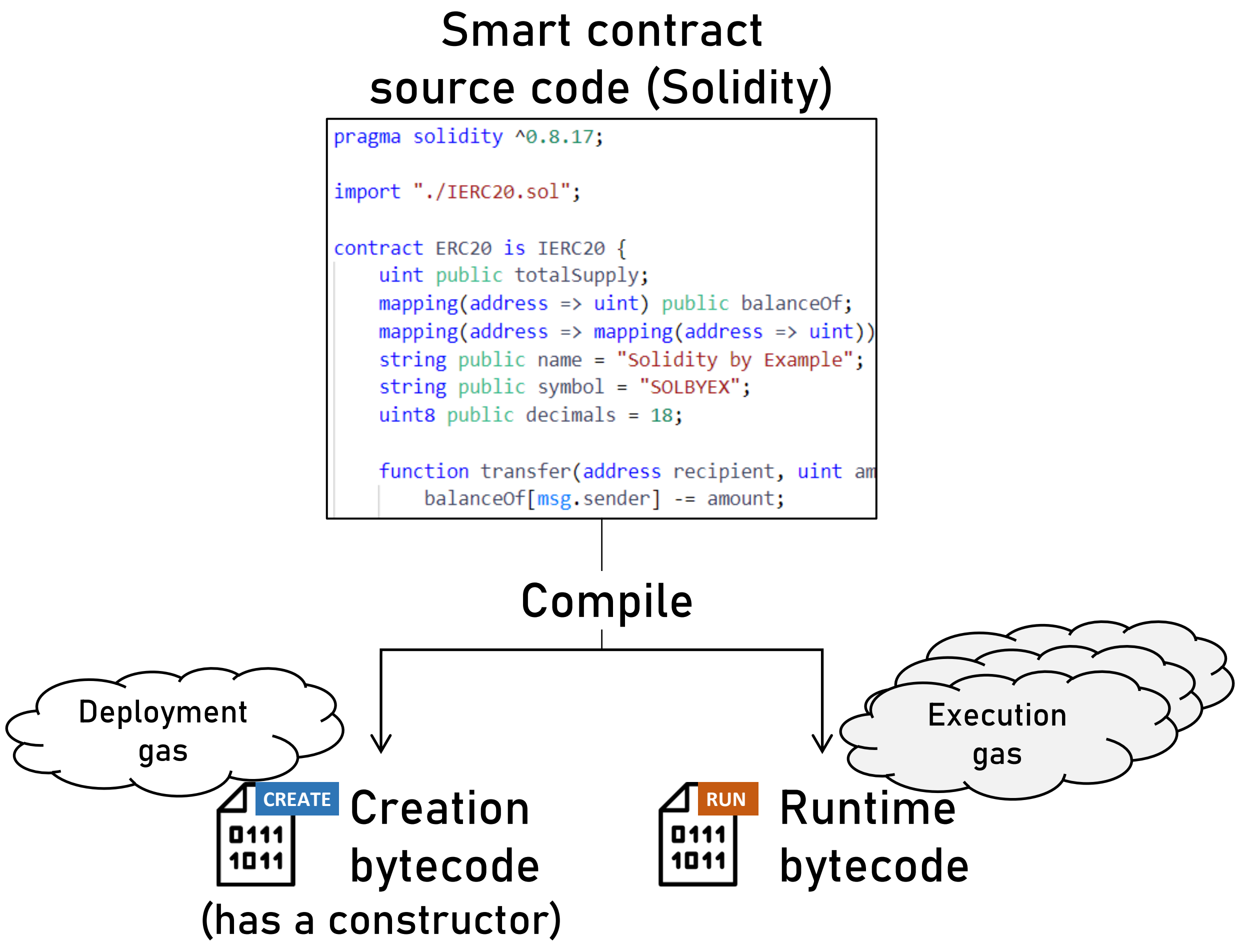
- ➢ Size and execution gas of a contract stay intact even after watermarking

② Adopt a **randomized approach** of electing watermark bytes from a contract bytecode

- ➢ An adversary cannot locate a watermark through static analyses

③ Insert the **hash** of the watermark location in a **creation bytecode (constructor)**

- ➢ The watermark location is confidential



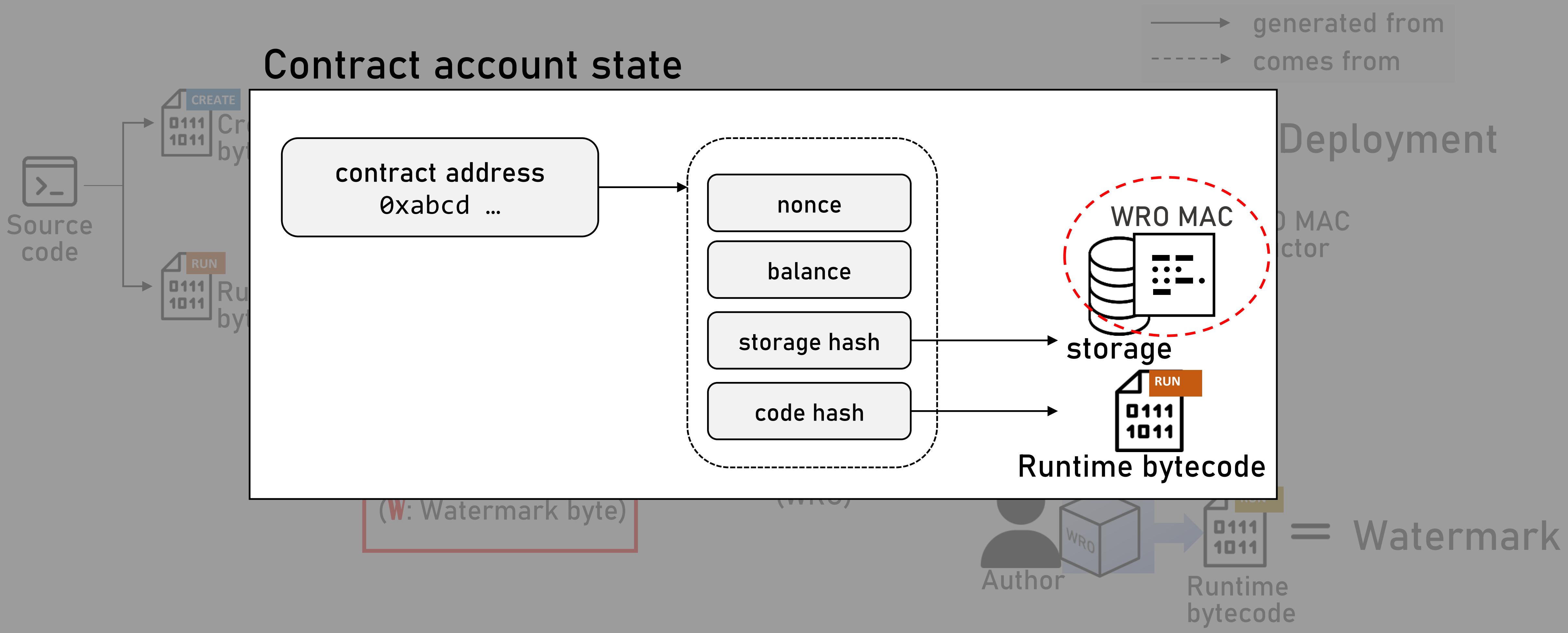
Smart contract source code (Solidity)

```
pragma solidity ^0.8.17;

import "./IERC20.sol";

contract ERC20 is IERC20 {
    uint public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint))
    string public name = "Solidity by Example";
    string public symbol = "SOLBYEX";
    uint8 public decimals = 18;

    function transfer(address recipient, uint am
        balanceOf[msg.sender] -= amount;
```

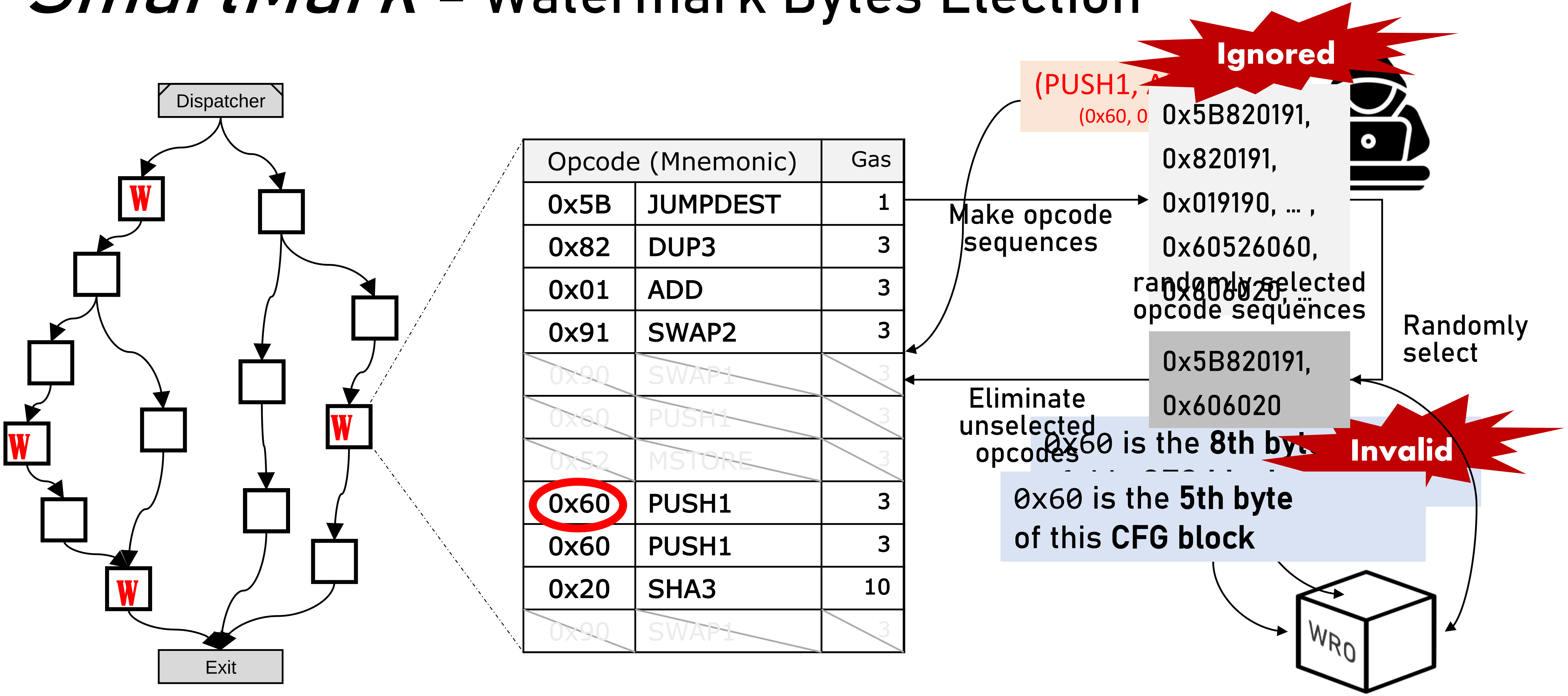
# *SmartMark* - Overview

- Our proposed software watermarking scheme for smart contracts

Contract account state



# *SmartMark* – Watermark Bytes Election



| Opcode (Mnemonic) | | Gas |
|---|---|---|
| 0x5B | JUMPDEST | 1 |
| 0x82 | DUP3 | 3 |
| 0x01 | ADD | 3 |
| 0x91 | SWAP2 | 3 |
| ~~0x90~~ | ~~SWAP1~~ | ~~3~~ |
| ~~0x60~~ | ~~PUSH1~~ | ~~3~~ |
| ~~0x52~~ | ~~MSTORE~~ | ~~3~~ |
| 0x60 | PUSH1 | 3 |
| 0x60 | PUSH1 | 3 |
| 0x20 | SHA3 | 10 |
| ~~0x90~~ | ~~SWAP1~~ | ~~3~~ |

0x5B820191,
0x820191,
0x019190, … ,
0x60526060,
0x606020, …

0x5B820191,
0x606020

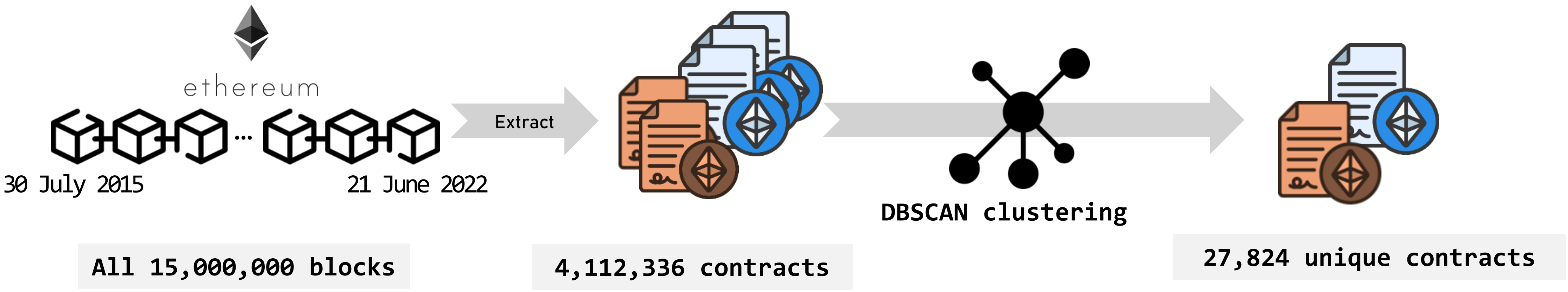
0x60 is the 8th byt
of this CFG block

0x60 is the 5th byte
of this CFG block

(W: Watermark byte)

# How Efficient *SmartMark* is?

- Collected all 15,000,000 blocks in the Ethereum Mainnet
- Obtained **27,824 unique contracts** using DBSCAN clustering from 4M smart contracts



ethereum

30 July 2015

21 June 2022

Extract

DBSCAN clustering

All 15,000,000 blocks

4,112,336 contracts

27,824 unique contracts

➢ In *SmartMark*, an embedding process and a verification process take average 11sec and 17sec, respectively, which is practically acceptable

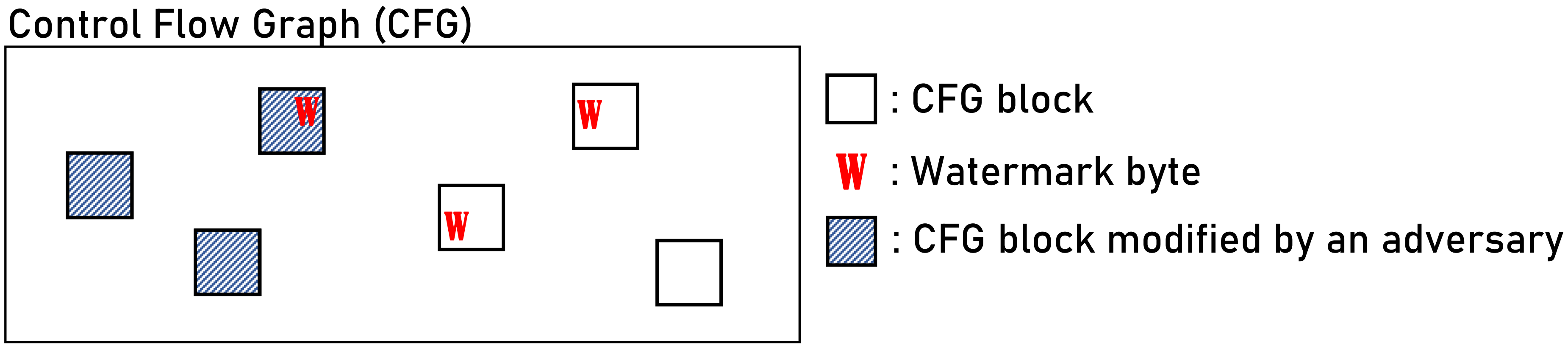
# How Robust *SmartMark* is?

- **Addition attack** embeds another watermark into an already watermarked contract and redeploys it
- **Deletion attack** eliminates a valid watermark from a watermarked contract
- **Distortion attack** encompasses transformation for damaging a watermark within a contract

➢ *SmartMark* is resilient to these three attacks that aim to corrupt a watermark

# Theoretical Analysis on Distortion Attacks

- The **attack success probability** of an adversary to successfully disable a watermark distorting a contract



➢ Only 8.9% of 27,824 contracts would be thwarted with more than 5% of attack success probability

$$P_{attack}(L, B_s, M_s) = \frac{\sum_{i=1}^{min(L,M_s)} \binom{B_s}{M_s}\binom{M_s}{i}\binom{B_s-M_s}{L-i}}{\binom{B_s}{L}\binom{B_s}{M_s}}$$

*L*: Length of a watermark
***B_s***: # Watermarkable blocks
***M_s***: # Watermarkable blocks modified by an adversary

# Conclusion

- We present *SmartMark*, a software watermarking scheme tailored to smart contracts
- We show *SmartMark*'s efficiency, effectiveness, and attack resiliency through our empirical results and theoretical analysis
- We publicly release *SmartMark* source code and experimental dataset*

* https://github.com/SKKU-SecLab/SmartMark.git

# Thank you, Any questions?

https://github.com/SKKU-SecLab/SmartMark

https://doi.org/10.6084/m9.figshare.21966875.v2