SADPonzi: Detecting and Characterizing Ponzi Schemes in Ethereum Smart Contracts

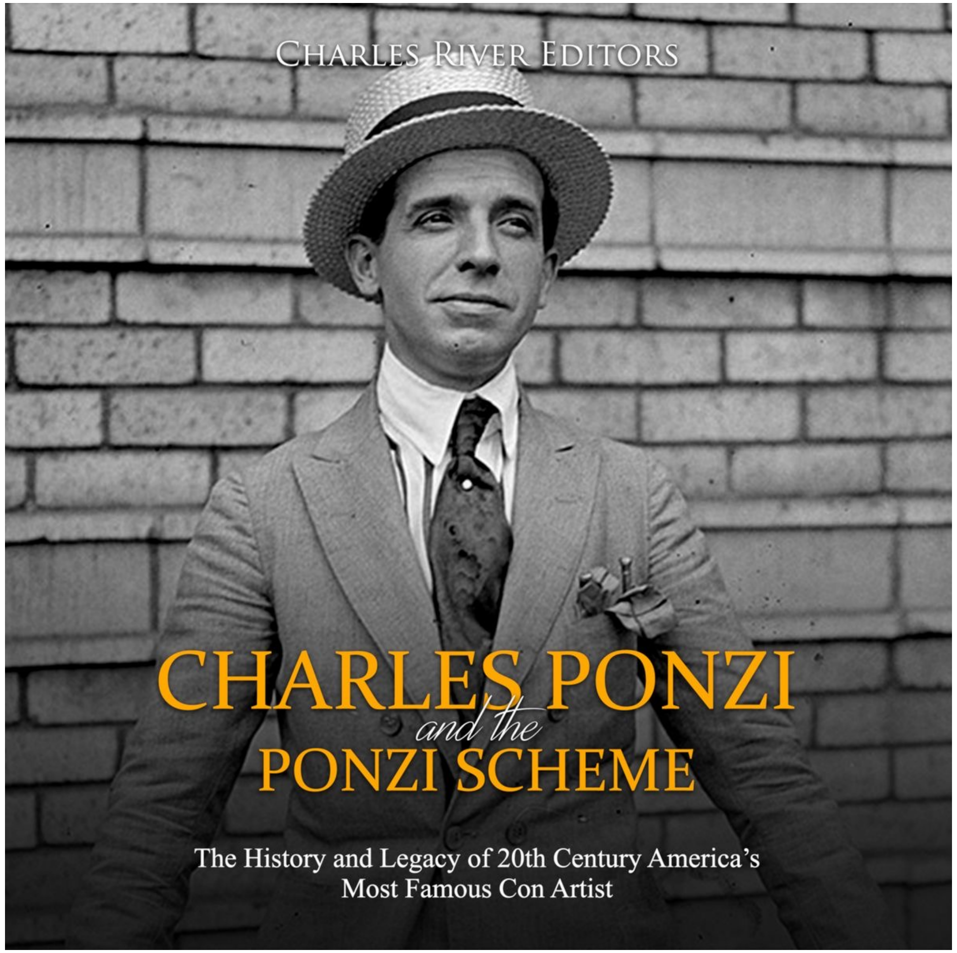
Presenter: Weimin Chen

WEIMIN CHEN, XINRAN LI, YUTING SUI,

NINGYU HE, HAOYU WANG*, LEI WU, XIAPU LUO

* corresponding author

Ponzi Scheme



Step by Step

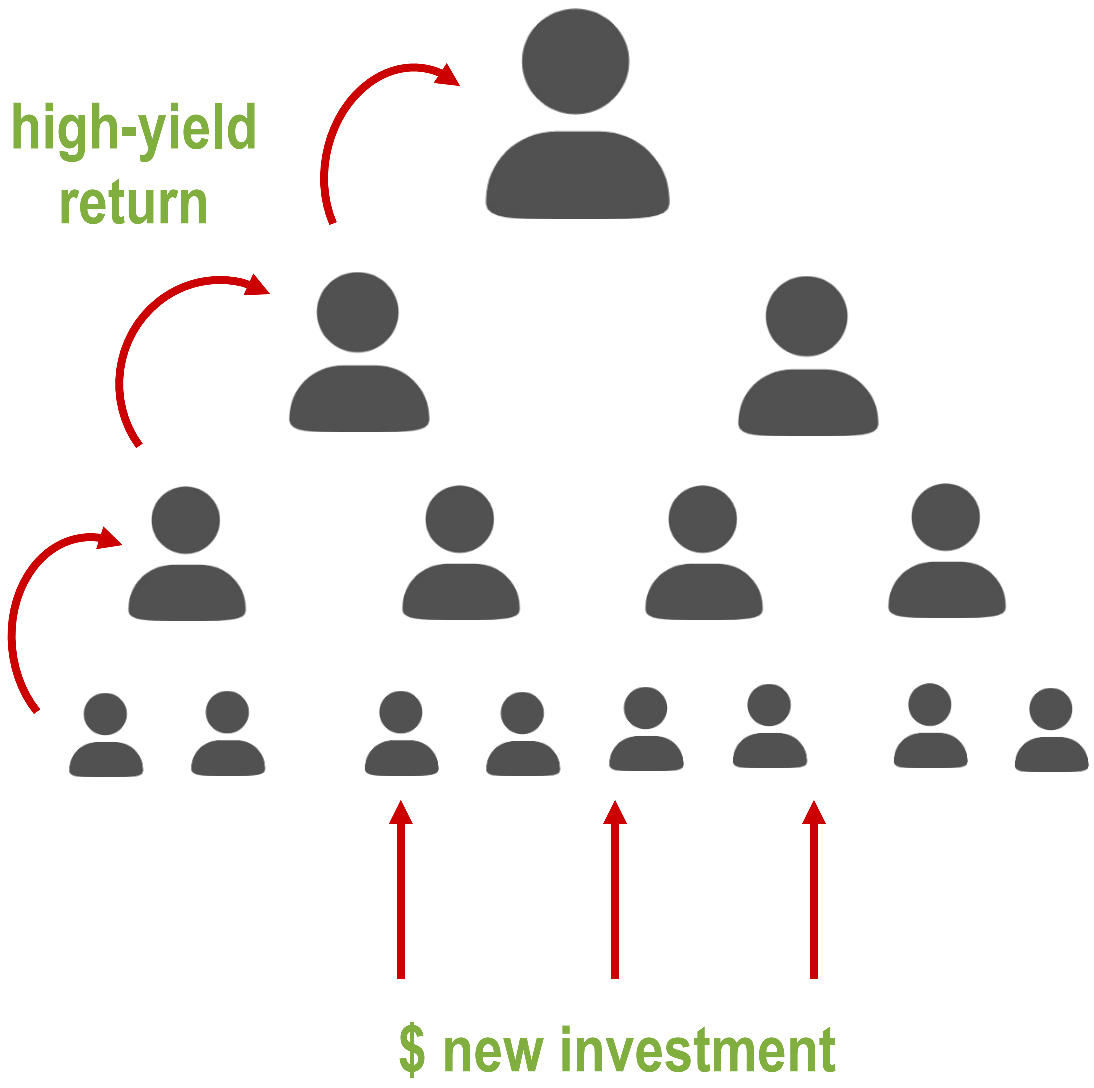
1. Promise the investors **a high-yield return**

2. Continue the scam by using part of **new investment** to reward to earlier investors.

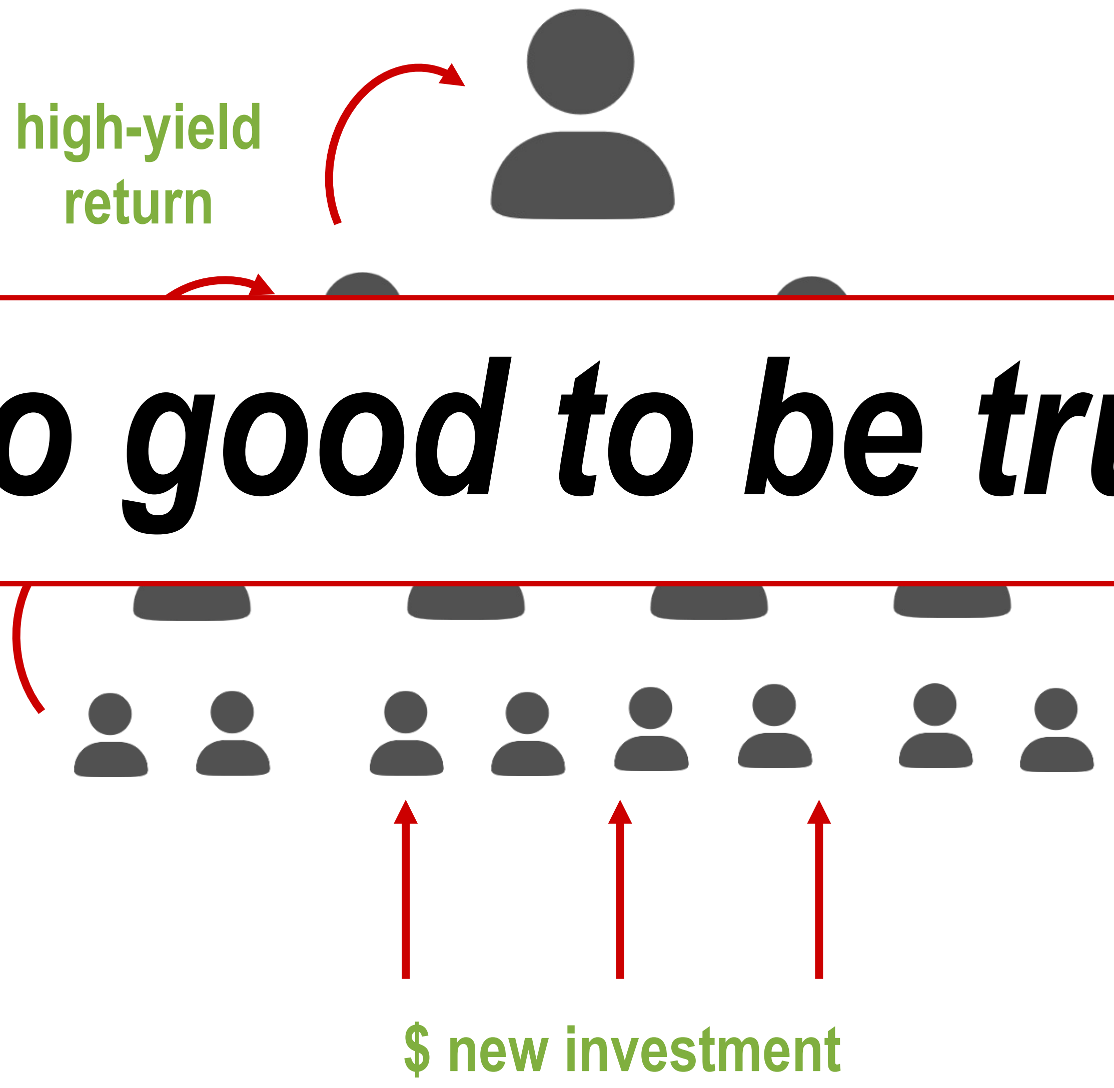
3. In return, the rewarded investors would help scam attract new investors.

4. Crash: when there are no enough new investments

Ponzi Scheme



Ponzi Scheme



Too good to be true !

Ponzi Scheme has emerged in the blockchain ecosystem

coindesk

Inside a Crypto ‘Ponzi’: How the $6.5M Banana.Fund Fraud Unravelled

Danny Nelson
August 3, 2020 · 5 min read

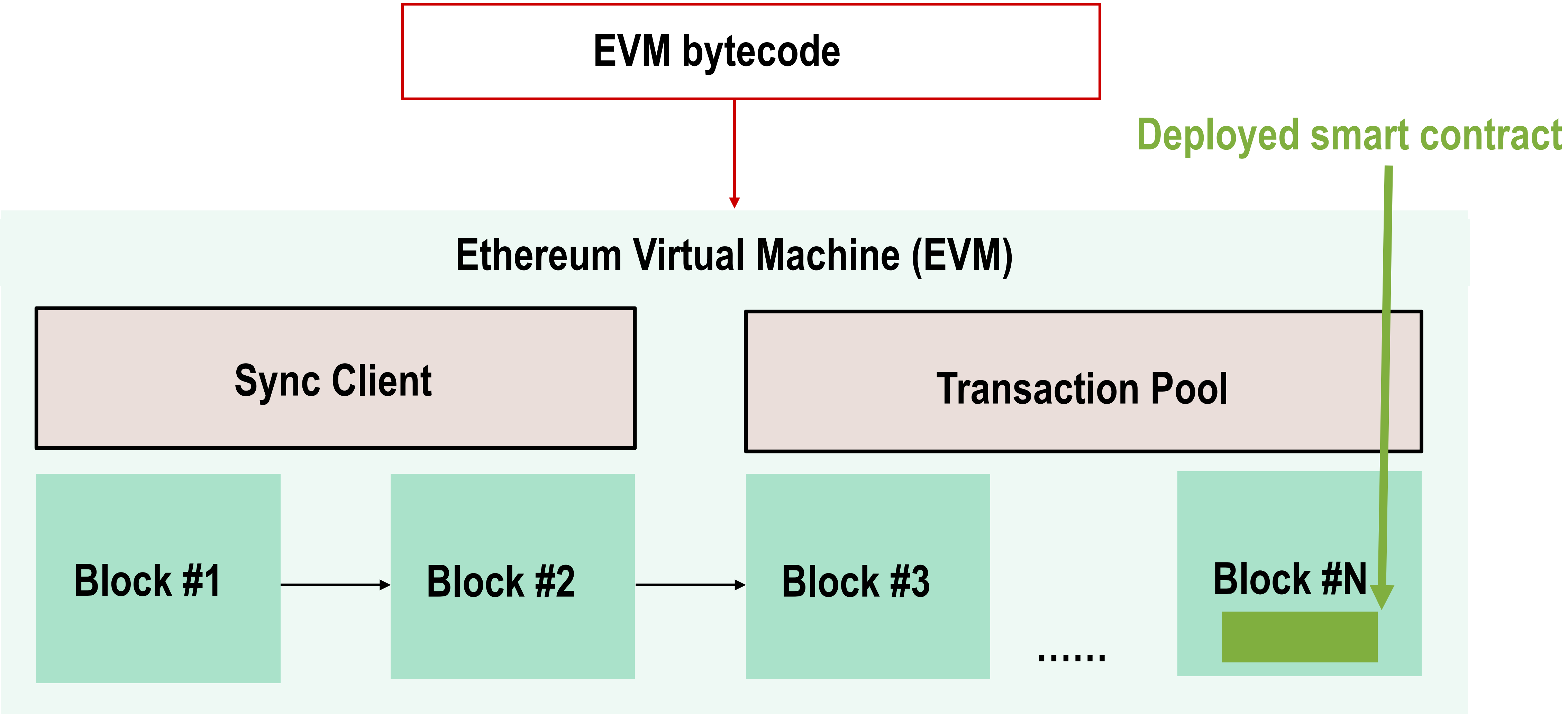
Quote Lookup

Related Quotes

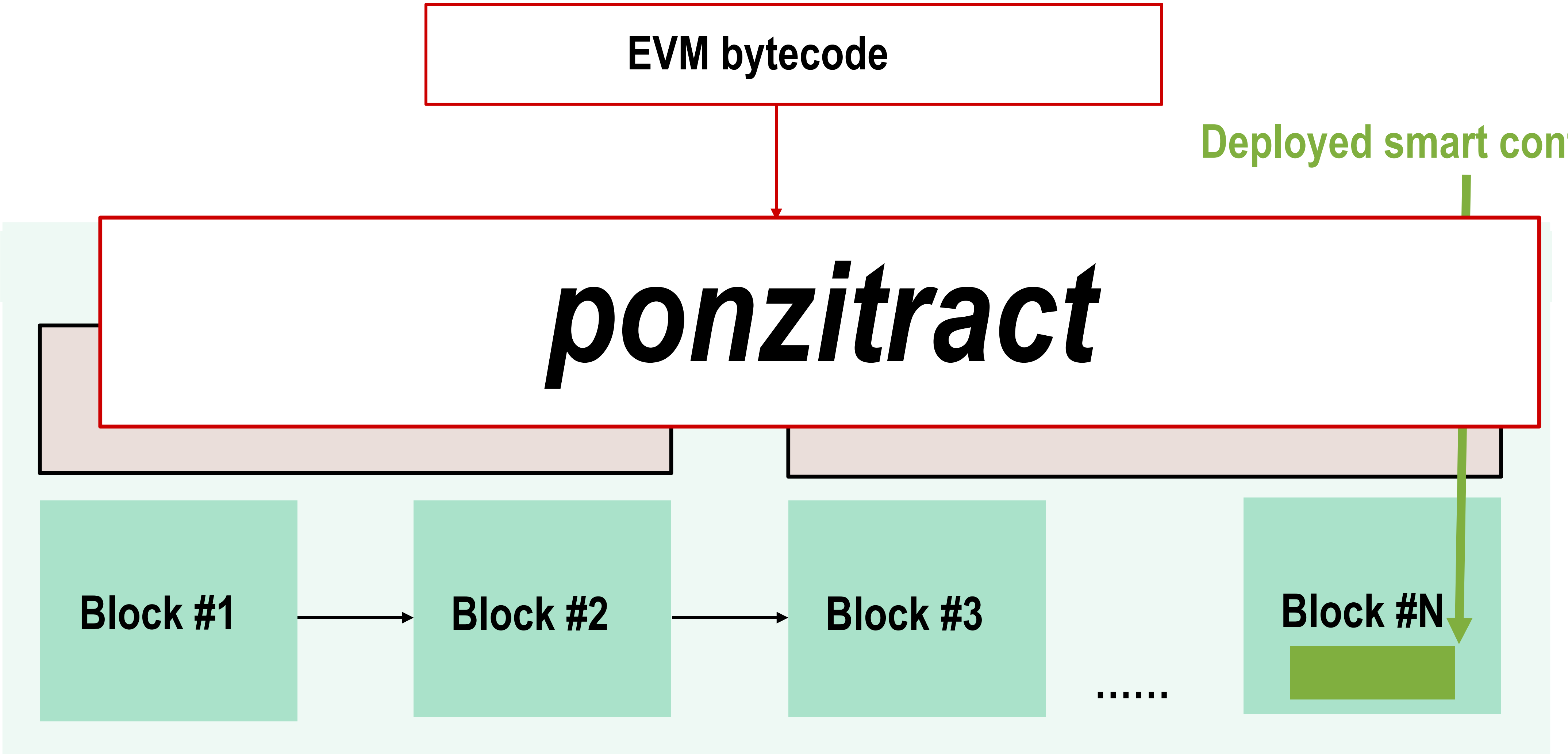
Ponzi Scheme has emerged in the blockchain ecosystem

Ponzi Scheme has emerged in the blockchain ecosystem

Ponzi Scheme has emerged in the blockchain ecosystem



Ponzi Scheme has emerged in the blockchain ecosystem



Ponzi Scheme has emerged in the blockchain ecosystem

What advantages be taken by the initiators of ponzitracts?

1. **Anonymous.**
   - No one can track the identity.
   - The initiator of a Ponzi scheme can stay anonymous.
2. **Enforce.**
   - No one can stop the execution of smart contracts.
   - Money extracted from the Ponzi Scheme do not need to be disclosed.
3. **Lost.**
   - No one can deny the transactions.
   - The victim's money cannot be found back.

Ponzi Scheme has emerged in the blockchain ecosystem

```
contract HandoverPonzi {
address public throne ;
uint public price = 1 ether;
uint fee = 0;
 function(){//fallback function
  if(msg.value < price) throw;
  fee += msg.value * 0.1;
  throne.transfer(msg.value*0.9);//reward
  throne = msg.sender;//invest
   price = price * 2;
}
```

Handover-scheme

```
contract ChainPonzi {
  uint balance = 0;
  uint Cursor = 0;
  struct Participant {
    address etherAddress;
    uint payout;
  }
  Participant[] participants;
  function() {
     participants.push(Participant
     (msg.sender, msg.value* 3));
     balance += msg.value ;
     while (balance > participants[Cursor].
                                payout) {
       uint payoutToSend = participants
                         [Cursor].payout;
       participants[Cursor].etherAddress
             .send(payoutToSend);
       balance = participants
                         [Cursor].payout;
       Cursor += 1;
     }
  }
}
```

Chain-scheme

```
contract TreePonzi {
  struct Participant {
    address inviter;
    address herself;
  }
  mapping(address => Participant) Tree;
  address top = 0xffff..ffff;
  function enter(address inviter) public {
    uint amount = msg.value;
     Tree[msg.sender] = Participant(
             {herself: msg.sender,
             inviter: inviter});
     address next = inviter;
     while (next != top) {
       amount /= 2;
       next.send(amount);
       next = Tree[next].inviter;
     }
  }
}
```

Tree-scheme

```
contract PonziICO {
  uint public total;
  mapping (address => uint)
                  public invested;
  mapping (address => uint)
                  public balanceOf;
  address[] investors;
  address owner = 0x4f22...1e;
  function withdraw() public{
    uint amount = balanceOf[msg.sender];
    balanceOf[msg.sender] = 0;
    msg.sender.transfer(amount);
  }

  function invest() private{
    uint dividend = msg.value;
    for(uint i=0;i<investors.length;i++) {
      uint amount = dividend * invested[in
                 vestors[i]] / total;
      balanceOf[investors[i]] += amount;
    }
    investors.push(msg.sender);
    invested[msg.sender] = msg.value;
    total += msg.value;
  }
  function() payable{ invest(); }
}
```

Withdraw-scheme

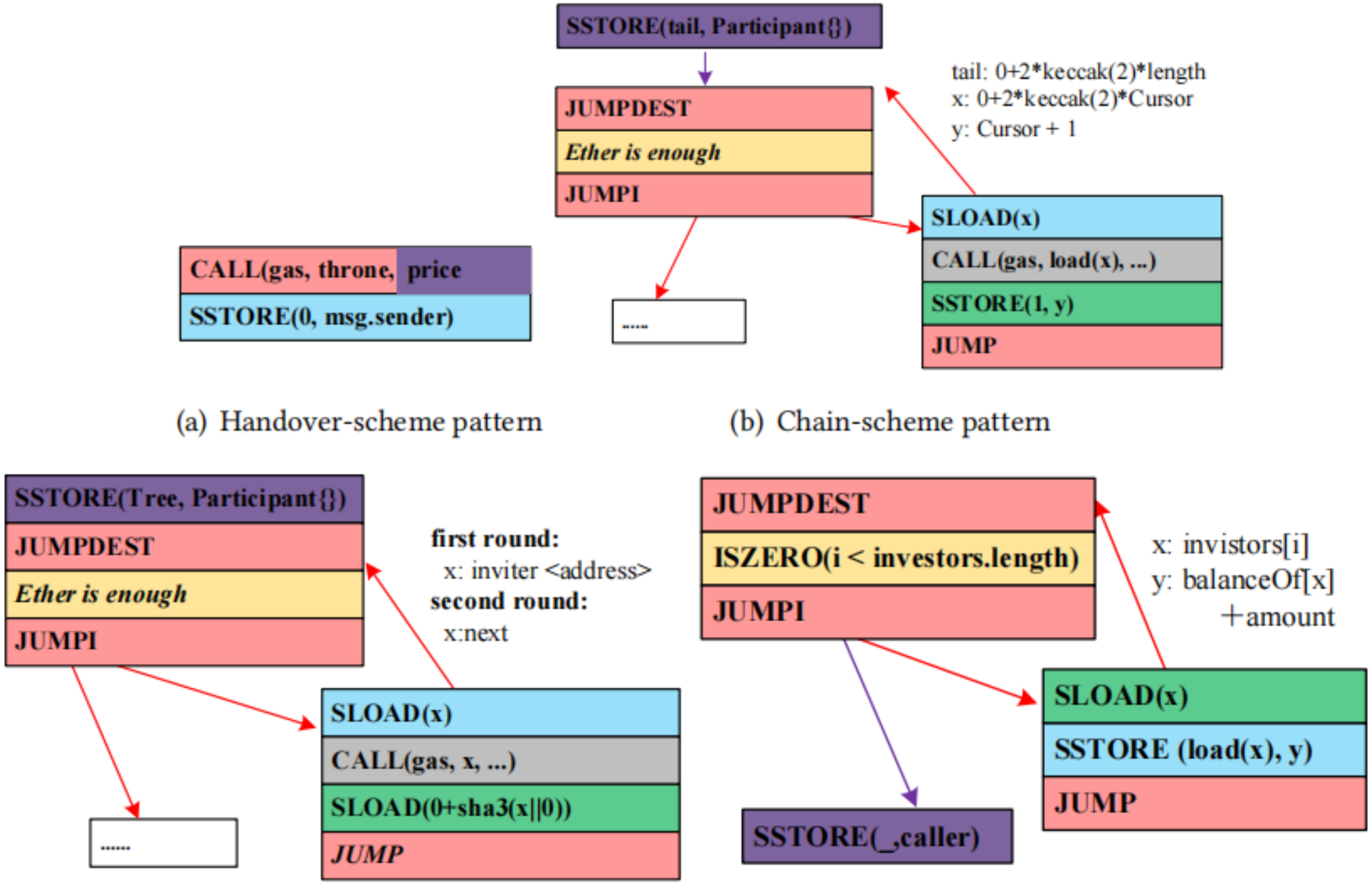
State-of-the-art

- **Survey:**
  - [55] presented the first empirical analysis of Bitcoin-based scams in 2015
- **Automatic tools:**
  - All existing efforts are taking advantage of machine learning techniques (**ML**)
  - *TxML [42]* relies on the **transaction behaviors** of smart contracts to perform detection. i.e., Gini coefficient [1], the inequality of returns to investors.
  - *OPCodeML [32]* takes the opcode extracted from the smart contract bytecode as features to train a classifier. The basic idea is that the **opcode frequency distribution** should differ between Ponzi and non-Ponzi smart contracts.

The limitations of state-of-the-art

- ***TxML [42]***
  - Poor scalability
  - Requires a considerable number of transactions to learn the behaviors.
  - Only popular ponzitracts that have lured a number of victims can be identified.
- ***OPCodeML [32]***
  - Lack of interpretability.
  - Be prone to evasion techniques, e.g., adding or removing some opcodes

Semantics of ponzitract



(a) Handover-scheme pattern (b) Chain-scheme pattern

(c) Tree-scheme pattern. (d) Withdraw-scheme pattern.

Semantics of ponzitract

Tree-scheme:

```
contract TreePonzi {
  struct Participant {
    address inviter;
    address herself;
  }
  mapping(address => Participant) Tree;
  address top = 0xffff..ffff;
  function enter(address inviter) public {
    uint amount = msg.value;
    Tree[msg.sender] = Participant(
           {herself: msg.sender,
           inviter: inviter});
    address next = inviter;
    while (next != top) {
      amount /= 2;
      next.send(amount);
      next = Tree[next].inviter;
    }
  }
}
```

the investor

investing action

rewarding action

the investment

Semantics of ponzitract

```
contract TreePonzi {
  struct Participant {
    address inviter;
    address herself;
  }
  mapping(address => Participant) Tree;
  address top = 0xffff..ffff;
  function enter(address inviter) public {
    uint amount = msg.value;
    Tree[msg.sender] = Participant(
        {herself: msg.sender,
        inviter: inviter});
    address next = inviter;
    while (next != top) {
      amount /= 2;
      next.send(amount);
      next = Tree[next].inviter;
    }
  }
}
```

Generate Semantic information with symbolic execution technology

Identify ponzitracts base on low-level bytecode

Challenge.1: Performing Internal Calls

- Invoking an internal call
  - CALL – transfer cryptocurrency, e.g., Ether
  - STATICCALL
  - DELEGATECALL
- Solution
  - Build cross-contract paths
  - Perform symbolic execution to recover ***return value***

Challenge.2: Parsing Storage Variables

- Persistent Data (Storage)
  - State data
  - Dynamical-size array (DSA)
  - Mapping data
- Storage Data Representation
  - *Storage* $[\sigma]$ $[\epsilon : \epsilon + o]$
  - $\theta\ (\sigma, \epsilon, o)$

```
contract PonziICO {
  uint public total;
  mapping (address => uint)
                     public invested;
  mapping (address => uint)
                     public balanceOf;
  address[] investors;
  address owner = 0x4f22...1e;
  function withdraw() public{
    uint amount = balanceOf[msg.sender];
    balanceOf[msg.sender] = 0;
    msg.sender.transfer(amount);
  }
```

source code

```
0x00 PUSH1    0x60
0x02 PUSH     0x40
0x04 MSTORE
0x05 PUSH     0xe0
...
0x10 PUSH1    0x4
0x11 SLOAD
0x12 PUSH1    0x9f
0x13 AND
...
```

bytecode

Storage[4][0:160]

semantics

Challenge.2: Parsing Storage Variables

- Z3 expression with abstract syntax tree (AST)
- Storage variables with address-value structure

SLOAD(4)
PUSH 159
AND

→ $Extract(160,0,Select(S,4))$, equivalent to $\theta(4,0,160)$

$$
\begin{aligned}
<func> &:= concat \mid extract \mid select \mid sha3 \\
<expr> &:= func(expr*) \mid expr_1\ [+-*/]\ expr_2 \\
<concat> &:= expr_1 \| expr_2 \\
<extract> &:= expr[p_h \ldots p_l] \\
<select> &:= S[expr] \\
<sha3> &:= sha3(expr)
\end{aligned}
$$

(a) Context-free syntax.

$$
\begin{aligned}
\mathrm{Var} &:= extract(\epsilon + o, \epsilon, select(S, \sigma)) \\
\sigma^{\mathrm{state}} &:= base \\
\sigma^{\mathrm{dsa}} &:= sha3(base) + width * key + offset \\
\sigma^{\mathrm{mapping}} &:= offset + sha3(concat(0, key, base))
\end{aligned}
$$

(b) Syntax for locating three important types.

Fig. 8. Syntaxes of the Storage Variables.

SADPonzi

- Overall architecture

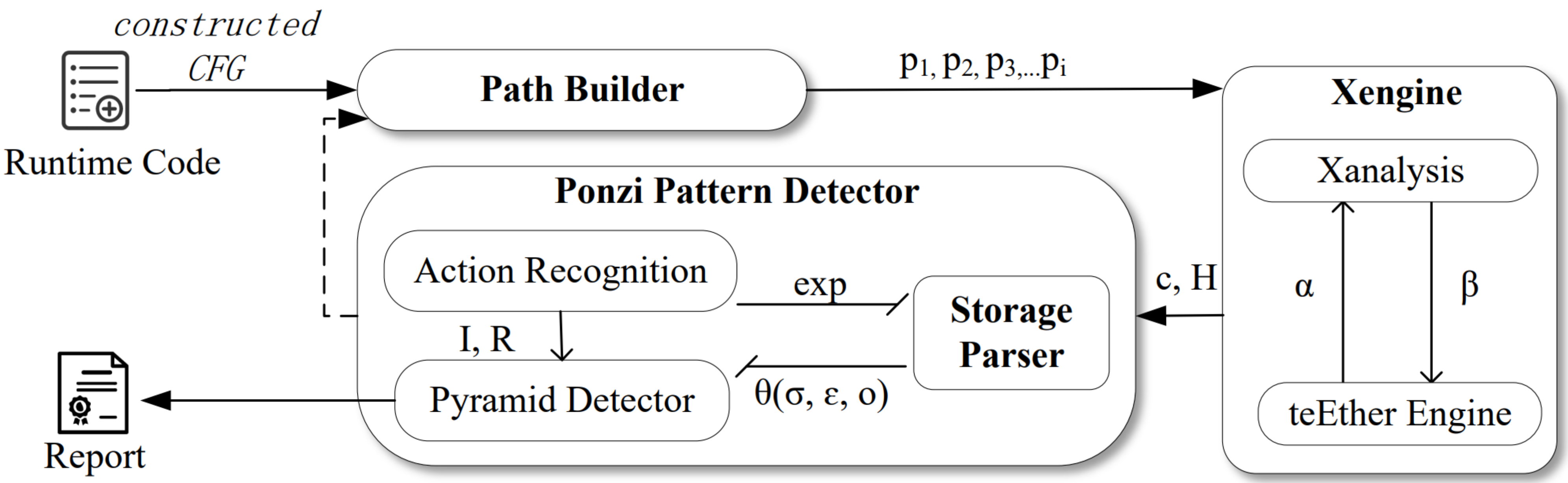
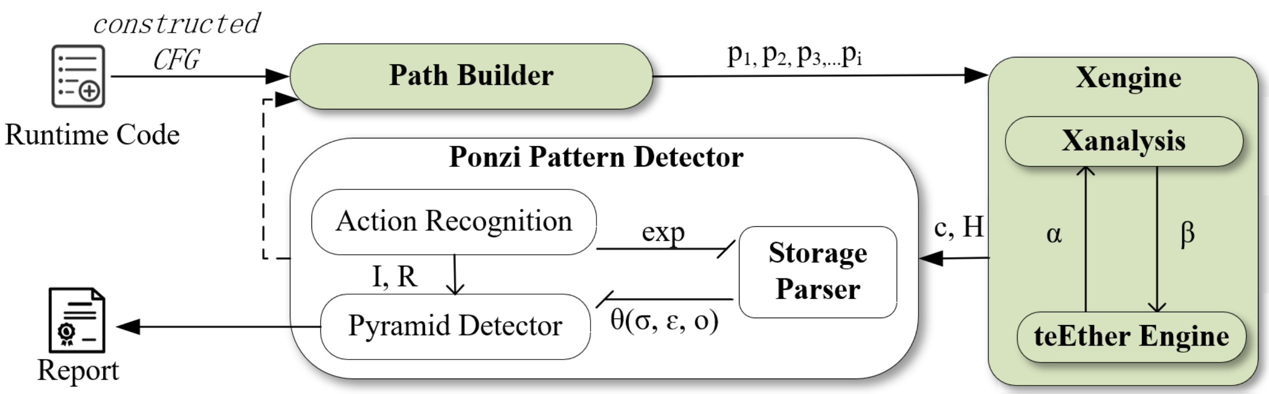


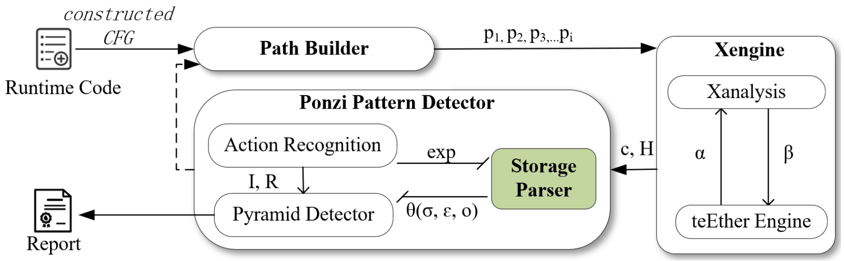
Fig. 7. The overall architecture of SADPonzi.

SADPonzi



- Path Builder
  - Generate concrete paths of **investing actions** and **reward actions**
  - Prioritize analyzing the opcodes of the fallback function
  - Propose heuristic strategies to limit the depth of the path
- Xengine
  - Build a symbolic context for each path
  - Based on teEther with Z3 as the SMT backend
  - Cross-contract analysis

SADPonzi



- Ponzi Pattern Detector-Storage Parser
  - Variable expression
  - Address semantics

    state, mapping and DSA variables
  - Algorithm1

    $Var := extract(o + \epsilon, \epsilon, S\,[\sigma])$

    ---> $\theta\ (\sigma, \epsilon, o)$

label a storage variable

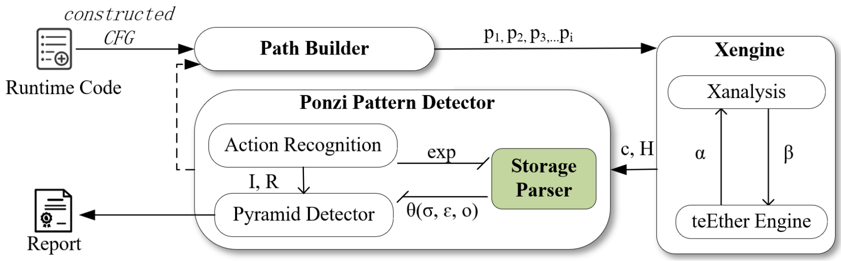
Algorithm 1 Getting the Slot of Storage Variable

Input: symbolic expression: expr
Output: θ.

```
if expr.decl ≠ extract then
    return
else:
    //expr = Extract(ϵ + o, o, S[σ])
    o, ϵ ← expr.params // tuple (size, shift)
    expr ← get_left_tree(expr)
    if expr.decl = select then
        // expr = S[σ]
        σ ← get_right_tree(expr)
        return σ, ϵ, o
```

get_left_tree(expr): get the left trees of expr.
get_right_tree(expr): get the right trees of expr.

SADPonzi



- Ponzi Pattern Detector-Storage Parser
  - Algorithm2
  - Recovers the raw parameters of σ
  - Recognizes the type of the variable

Algorithm 2 Storage Variables Recognition

Input: the symbolic expression of slot: σ
Output: type of storage variable and the tuple of σ.

```
if type(σ) = concrete then
    return tag1, (σ)
else
    items ← get_trees(σ)
    if len(items) = 3 then
        // DSA @ hash + width*key + offset
        base ← get_hash_idx(items[0])
        assert (items[1].decl = bvmul)
        width ← get_left_tree(items[1])
        key ← get_right_tree(items[1])
        offset ← item[2]
        return tag2, (base, key, width, offset)
    else:
        offset, expr ← items
        if exp ∈ H then
            // mapping @ offset + sha3(key||base)
            identitiy ← H[exp] | get_hash_idx(expr)
            assert (identitiy.decl = concat)
            key, base ← identity[: 256], identity[256 :]
            // removing padding
            if key.decl = concat then
                key ← get_right_tree(key)
            if base.decl = concat then
                base ← get_right_tree(base)
            return tag3, (base, key, offset)
```

get_trees(expr): get the sub trees of input.
get_hash_idx(hash): get the concrete input of the keccak256. computation according the hash value if possible.

$$\mathrm{Var} := extract(\epsilon + o, \epsilon, select(S, \sigma))$$

$$\sigma^{\mathrm{state}} := base$$

$$\sigma^{\mathrm{dsa}} := sha3(base) + width * key + offset$$

$$\sigma^{\mathrm{mapping}} := offset + sha3(concat(0, key, base))$$

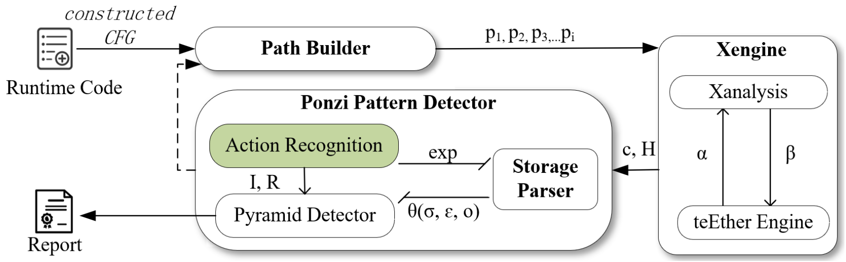
(b) Syntax for locating three important types.

SADPonzi

A ponzitract contains a redistribution scenario to divide new investments among its previous participants, and thus the money flow is pyramid-shape.

- **Ponzi Pattern Detector-Summary of Semantics**
  - 1) **Investing action (I):** a player invokes a transaction to invest Ether, and the ponzitract stores user information.
  - 2) **Rewarding action (R):** the ponzitract pays a bonus for the participated players.
  - 3) A **pyramid distribution strategy** is associated with the achievement of I and R.
    - e.g., Handover, Tree-shape, Chain-shape, Withdraw-shape

SADPonzi

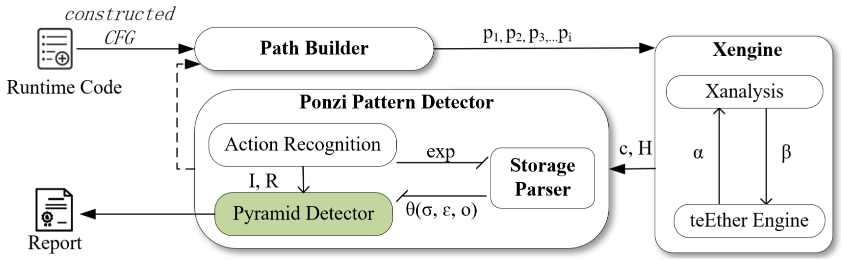


- Ponzi Pattern Detector-Action Recognition
  - Investing action

$$i_r = \mathsf{SSTORE}_1(\_, \tau_1) \land is\_r(\tau_1) \land i_v = \mathsf{SSTORE}_2(\_, \tau_2) \land is\_v(\tau_2)$$
$$\land (\{\mu = S[*][8 * i : 160 + 8 * i] \mid i \in \mathbb{N}, i \leq 12\}) \not\subseteq c \land ((\lambda > 0 \in c) \lor f \notin sigs)$$

  - Rewarding action

$$i = \mathsf{CALL}(\_, t, b) \land ((t = I_r) \lor ((t = \mu) \land (b = I_v)))$$



SADPonzi

- Ponzi Pattern Detector-Pyramid Detector
  - Handover-scheme

$$tag(I_r) = tag_1 \wedge I_r = R_t \wedge bb(I_{i_r}) \in dcs(bb(R_i))$$

  - Chain-scheme

$$tag(I_r) = tag_2 \wedge I_r = R_t$$

$$\wedge\ (tag(R_t^{key}) = tag_1 \vee (type(R_t^{key}) = \text{integer} \wedge is\_cycle(\mathcal{G}, bb(R_i))))$$

  - Tree-scheme

$$tag(I_r) = tag_3 \wedge I_r = R_t \wedge I_r = \theta_{I_r}(\_, \_, 160)$$

  - Withdraw-scheme

$$R_r = \mu \wedge tag(R_b) = tag_2 \wedge I_v = R_b \wedge is\_cycle(\mathcal{G}, bb(I_{i_v}))$$

RQ #1: *The Effectiveness of SADPonzi*

- **Benchmark: 1,395 samples**
  - 1,262 non-Ponzi samples from DAppTotal
  - 133 Ponzi contracts remarked from M. Bartoletti (removed duplicated contracts and Non-Ponzi contracts)
- **Experimental Setup**
  - Performing a server running Ubutnu 18.04 (i9-9900 CPU and 64 GB RAM)
  - Setting time-out as 20 minutes and 30 layers as maximum call depth
- **Baseline：10-fold cross validation**
  - TxML：analyzing some features of on-chain transactions (RandomForest)
  - OPCodeML：analyzing the frequency of opcode sequence (XGBoost)

RQ #1: *The Effectiveness of SADPonzi*

Table 3. Overall Evaluation Results on the Benchmark.

Approach	Precision	Recall	F1-measure
TxML [42]	76.7%	57.9%	66.0%
OPCodeML [32]	92.0%	85.2%	88.5%
SADPonzi	**100%**	**100%**	**100%**

- The result of *TxML* is inline with the evaluation in previous work
- *OPCodeML* performs better than the one reported in their paper

SADPonzi outperforms existing state-of-the-art machine learning approaches. SADPonzi, preserving full semantic information of contract codes, achieves a good balance between accuracy and scalability

RQ #2: *The Robustness of SADPonzi*

Table 4. A Comparison of the Robustness of SADPonzi and OPCodeML.

	# Recomplie		# LAO		# DFO		# LDO	
	FP	FN	FP	FN	FP	FN	FP	FN
OPCodeML	11/595	73/76	0/321	74/76	0/69	64/65	1/50	60/62
SADPonzi	0/595	**0/76***	0/321	**0/76***	0/69	**0/65**	0/50	**0/62**

* SADPonzi reports correctly as long as given larger computational resources.

- BiAn obfuscates source code
  - Layout Obfuscation (LAO)
  - Data flow Obfuscation (DFO)
  - Control Flow Obfuscation (CFO)

SADPonzi is resilient to four types of obfuscation strategies that are commonly used today. However, the performance of OPCodeML decreases sharply when facing obfuscated and evasive smart contracts.

RQ #3: *The Prevalence of Ponzi Smart Contracts*

- **Large-scale Dataset**

  83,269 unique contracts in total from July 1st, 2015 to May 20th, 2020

- **Overall Results**
  - *616 unique contracts (extended to 835)*
  - *56.05% (468/835) are Chain-scheme*

Table 5. The transactions related to Ponzi Smart Contracts.

Type	#Distinct Num	#Extend Num	Investment # Victim	# Transaction	# Ether	USD
Chain	344	468	3,120	227,649	47,228.47	8,330,629.8233
Tree	212	268	14,565	99,612	52,073.8385	9,185,304.3730
Withdraw	13	19	174	824	463.8878	81,825.1690
Handover	47	80	148	2,603	624.6784	110,187.0230
Total	**616**	**835**	**17,870**	**330,688**	**100,390.87**	**17,707,946.38**

RQ #3: *The Prevalence of Ponzi Smart Contracts*

- Creation time of ponzitracts
- Creators of ponzitracts

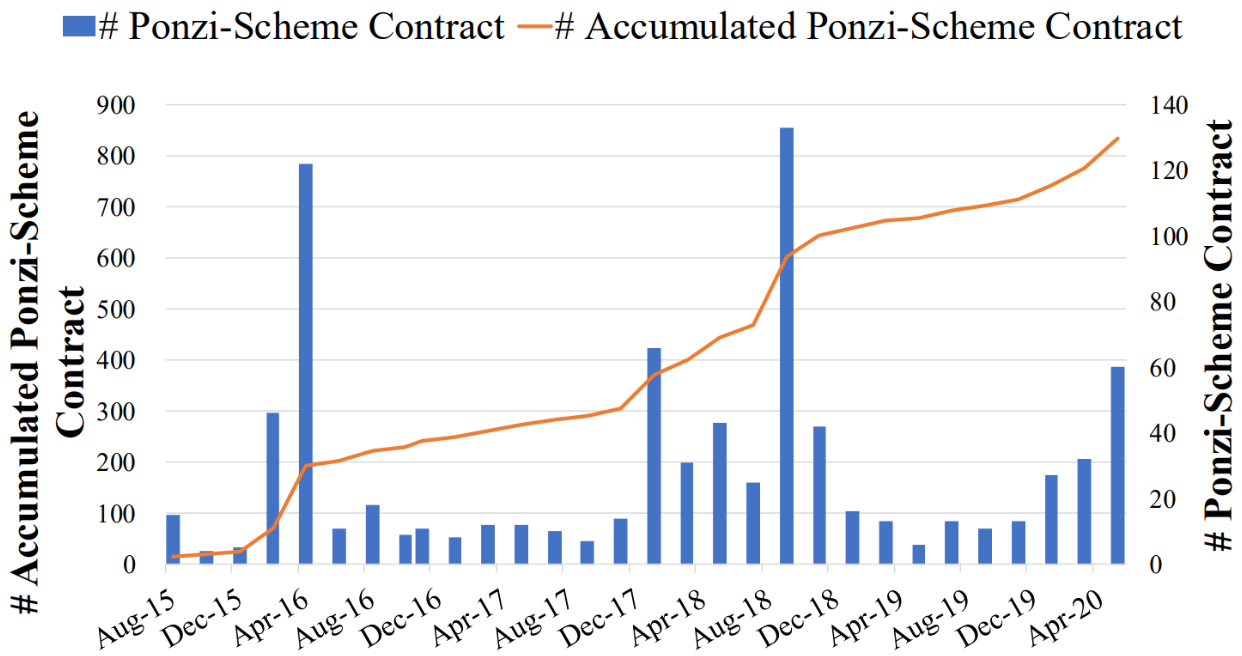


Fig. 9. The monthly distribution of ponzitracts.

We have identified 835 Ponzi schemes created by 444 EOAs in total. These Ponzi contracts were created all the time within the span of 5 years, peaking in April 2016 and October 2018. This suggests that Ponzi schemes are prevalent in the ecosystem.

RQ #4: *The Impact of Ponzi Smart Contracts*

- **The number of transactions**
  - Over **598.1 K** transaction records
  - **330.7 K** incoming transactions (**396** on average)
- **The number of victims and the amount of money**
  - **17 K** victims have sent over **100 K** ETH (over 17 million US Dollar)

Our observation suggests the great impact of ponzitracts, i.e., thousands of victims were scammed of millions of US Dollars. It reveals the urgency to identify and mitigate blockchain Ponzi schemes.

Ether Price:using the closing price of Ether on October 2019

Conclusion

- Propose a semantic-aware approach for ponzitract detection
- Develop SADPonzi working on Ethereum smart contract bytecode; Outperforms all ML baselines.
- Apply SADPonzi to all 3.4 million smart contracts on Ethereum; identify 835 ponzitracts involving over $ 17 Million

Thanks for Watching!

Q&A