Working Paper presented at the

Peer-to-Peer Financial Systems 2017 Workshop

2017

Dissecting Ponzi schemes on Ethereum: identification, analysis, and impact

Massimo Bartoletti
Università di Cagliari

Salvatore Carta
Università di Cagliari

Tiziana Cimoli
Università di Cagliari

Roberto Saia
Università di Cagliari

P2P Financial Systems

Powered by Exponential Science

Dissecting Ponzi schemes on Ethereum: identification, analysis, and impact

Massimo Bartoletti, Salvatore Carta, Tiziana Cimoli, Roberto Saia

Dipartimento di Matematica e Informatica - Università di Cagliari
Via Ospedale 72 - 09124 Cagliari, Italy
{bart,salvatore,t.cimoli,roberto.saia}@unica.it

Abstract. Ponzi schemes are financial frauds where, under the promise of high profits, users put their money, recovering their investment and interests only if enough users after them continue to invest money. Originated in the offline world 150 years ago, Ponzi schemes have since then migrated to the digital world, approaching first on the Web, and more recently hanging over cryptocurrencies like Bitcoin. Smart contract platforms like Ethereum have provided a new opportunity for scammers, who have now the possibility of creating "trustworthy" frauds that still make users lose money, but at least are guaranteed to execute "correctly". We present a comprehensive survey of Ponzi schemes on Ethereum, analysing their behaviour and their impact from various viewpoints. Perhaps surprisingly, we identify a remarkably high number of Ponzi schemes, despite the hosting platform has been operating for less than two years.

1 Introduction

The advent of Bitcoin [3, 12] has given birth to a new way to exchange currency, allowing secure and (almost) anonymous transfers of money without the intermediation of trusted authorities. This has been possible by suitably combining several techniques, among which digital signature schemes, moderately hard "proof-of-work" puzzles, and the idea of *blockchain*, an immutable public ledger which records all the money transactions, and is maintained by a peer-to-peer network through a distributed consensus protocol.

Soon after Bitcoin has become widespread, it has started arousing the interest of criminals, eager to find new ways to transfer currency without being tracked by investigators and surveillance authorities [9].

Recently, *Ponzi schemes* [1] — a classic fraud originated in the offline world at least 150 years ago — have approached the digital world, first on the Web [10], and more recently also on Bitcoin [16]. Ponzi schemes[1] are often disguised as "high-yield" investment programs. Users enter the scheme by investing some money. The actual conditions which allow to gain money depend on the specific rules of the scheme, but all Ponzi schemes have in common that, to redeem their

[1] They are named so after Charles Ponzi, a notorious fraudster from the 1920s.

investment, one has to make new users enter the scheme. A more authoritative definition of Ponzi schemes comes from the U.S. Securities and Exchange Commission (SEC):[2]

> "*A Ponzi scheme is an investment fraud that involves the payment of purported returns to existing investors from funds contributed by new investors. Ponzi scheme organizers often solicit new investors by promising to invest funds in opportunities claimed to generate high returns with little or no risk. With little or no legitimate earnings, Ponzi schemes require a constant flow of money from new investors to continue. Ponzi schemes inevitably collapse, most often when it becomes difficult to recruit new investors or when a large number of investors ask for their funds to be returned.* "

Often, the investment mechanism of Ponzi schemes creates a pyramid-shape topology of users, having at the top level the initiator of the scheme, and at level $\ell + 1$ the users who compensate the investment of those at level ℓ. The scheme will eventually collapse because at some point it will no longer be possible to find new investors, as their number grows exponentially in the number of levels of this pyramid. Therefore, users at the top levels of the pyramid will gain money, while those at the bottom layers will lose their investment.

Despite many investors are perfectly conscious of the fraudulent nature of these schemes, and of the fact that they are illegal in many countries, Ponzi schemes continue to attract remarkable amounts of money. A recent study [16] estimates that Ponzi schemes operated through Bitcoin have gathered more than 7 millions *USD* in the period from September 2013 to September 2014[3]. We expect that, since the total capitalization of Bitcoin has grown substantially since then[4], also the impact of Ponzi schemes has increased proportionally.

"Smart" Ponzi schemes. The spread of *smart contracts*, i.e., computer programs whose correct execution is automatically enforced without relying on a trusted authority [15], creates new opportunities for fraudsters. Indeed, implementing Ponzi schemes as smart contracts would have several attractive features:

1. The initiator of a Ponzi scheme could stay anonymous, since creating the contract and withdrawing money from it do not require to reveal his identity;
2. Since smart contracts are "unmodifiable" and "unstoppable", no central authority (in particular, no court of law) would be able to terminate the execution of the scheme, or revert its effects in order to refund the victims. This is particularly true for smart contracts running on *permissionless* blockchains, which are controlled by a peer-to-peer network of miners.
3. Investors may gain a false sense of trustworthiness from the fact that the code of smart contracts is public and immutable, and their execution is

[2] Source: www.sec.gov/spotlight/enf-actions-ponzi

[3] This estimate considers both traditional Ponzi schemes which also accept payments in bitcoins, and schemes that only handle bitcoins.

[4] The market capitalization of Bitcoin has grown from ~ 5 billions *USD* (9/2014) to 25 billion *USD* (5/2017). Source: coinmarketcap.com/currencies/bitcoin

automatically enforced. This may lead investors to believe that the owner cannot take advantage of their money, that the scheme would run forever, and that they have a fair probability of gaining the declared interests.

All these features are made possible by a combination of factors, among which the growth of platforms for smart contracts [13], which advertise anonymity and contract persistence as main selling points, and the fact that these technologies are very recent, and still live in a gray area of legal systems [7, 11].

Understanding the behaviour and impact of "smart" Ponzi schemes would be crucial to devise suitable intervention policies. To this purpose, one has to analyse various aspects of the fraud, answering to several questions: how many victims are involved? How much money is invested? What are the temporal evolution and the lifetime of a fraud? What kind of users fall in these frauds? Can we recognize fingerprints of Ponzi schemes during their execution, or possibly even before they are started? Investigating on these issues would help to disrupt this kind of frauds.

Contributions. This paper is the first comprehensive survey on Ponzi schemes in Ethereum [5], the most prominent platform for smart contracts so far[5]. We construct a collection of Ponzi schemes, we analyse them, and we measure their impact on Ethereum from various perspectives. More specifically:

- in Section 3 we present a methodology for collecting Ponzi schemes from the Ethereum blockchain. Basically, we start by examining all the contracts with a certified source code[6]. By inspecting it and by performing targeted searches on Google, we determine if the contract implements a Ponzi scheme. We expand our collection by searching the blockchain for contracts whose bytecode is highly similar to a contract already classified as a Ponzi scheme. In this way we collect a total of 191 Ponzi schemes. For each of them, we extract from the Ethereum blockchain all the related transactions, which record all the incoming and outgoing movements of money.
- in Section 4 we analyse the source code of Ponzi schemes for which it is available. We discover that most contracts share a few common patterns, and that many of them are obtained by minor variations of already existing ones. We show that the vast majority of the analysed contracts contain security vulnerabilities, which could be exploited by adversaries to steal money. We then analyse the way these contracts are advertised on the web, in many cases finding a discrepancy between the advertised chances to get a payout, and the actual ones.
- in Section 5 we compare the overall number of transactions related to Ponzi schemes against the transactions in the whole Ethereum blockchain. We also

[5] The market capitalization of Ethereum is more than 8 billion *USD* on 5/2017. Source: `coinmarketcap.com/currencies/ethereum`

[6] Even though only the contract bytecode is stored on the Ethereum blockchain, some explorers maintain also the source code, and certify that the bytecode of a contract is actually the result of the compilation of the associated source code.

measure the economic impact of scams, by quantifying the overall value in *USD* exchanged through Ponzi schemes.

- in Section 6 we investigate the lifetime of Ponzi schemes. This may be an important indicator to predict when a scheme is going to collapse.
- in Section 7 we focus on the top 10 schemes (those with the highest number of transactions), and for each of them we measure the gains and losses of users. In many cases we observe the typical pattern of Ponzi schemes: a few users gain a lot, while the majority of users simply lose their money.
- in Section 8 we study the temporal behaviour of Ponzi schemes, measuring the monthly volume of their transactions. We find that the volume of outgoing payments dominates that of incoming ones: this means that users invest more than the money they obtain back.
- in Section 9 we study what kind of users invest money in Ponzi schemes, measuring the inequality of payments to and from the schemes. This indicator may reveal how scammers select their victims: a fair distribution of payments means that the scheme is fed by a large number of victims who pay small amounts of money; instead, an unequal distribution often means that the scheme profits from a small number of "big fishes" who invest a lot of money.

Our dataset of Ponzi schemes is available online at `goo.gl/CvdxBp`.

2 Ethereum in a nutshell

Ethereum [5] is a decentralized virtual machine, which can execute programs, called *contracts*, written in a Turing-complete bytecode language [18]. Every contract has a permanent storage where to keep data, and a set of functions which can be invoked either by users or by other contracts. Users and contracts can own *ether*, a cryptocurrency similar to Bitcoin (denoted with *ETH*), and send/receive it to/from users or other contracts.

Users can send *transactions* to the Etherum network in order to: (i) create new contracts; (ii) invoke a function of a contract; (iii) transfer ether to contracts or to other users. All the transactions sent by users, called *external* transactions, are recorded on a public, append-only data structure — the *blockchain*. Upon receiving an external transaction, a contract can fire some *internal* transactions, which are not explictly recorded on the blockchain, but still have effects on the balance of users and of other contracts.

Since transactions can move money, it is crucial to guarantee that their execution is performed correctly. To this purpose, Ethereum does *not* rely on a trusted central authority: rather, each transaction is processed by a large network of mutually untrusted peers — called *miners*. Each miner executes all the transactions, and its results must match with those of the other miners. There is a *consensus* protocol to address mismatches (due e.g., to failures or to attacks), which is based on a "proof-of-work" puzzle. The security of the consensus protocol relies on the fact that, for a miner, following the protocol is

```
1  contract AWallet{
2      address owner;
3      mapping (address => uint) public outflow;
4      mapping (address => uint) public inflow;
5
6      function AWallet(){ owner = msg.sender; }
7
8      function pay(uint amount, address recipient) returns (bool){
9          if (msg.sender != owner ||  msg.value != 0) throw;
10         if (amount > this.balance) return false;
11         outflow[recipient] += amount;
12         if (!recipient.send(amount)) throw;
13         return true;
14     }
15
16     function(){ inflow[msg.sender] += msg.value; }
17 }
```

Fig. 1: A simple wallet contract.

more convenient than trying to attack it. Indeed, miners receive economic incentives for correctly performing all the computations required by the protocol. Incorrect computations are soon discovered (since they are not the majority), and discharged. Hence, the execution of contracts is guaranteed to be correct, as long as the adversary does not control the majority of the computational power of the network.

Ethereum smart contracts. Ethereum contracts are composed by fields and functions. A user can invoke a function by sending a suitable transaction to the Ethereum nodes. The transaction *must* include the execution fee (for the miners), and *may* include a transfer of ether from the caller to the contract.

We illustrate contracts through a small example (`AWallet`, in Figure 1), which implements a personal wallet associated to an owner. Rather than programming it directly as EVM bytecode, we use *Solidity*, a Javascript-like programming language which compiles into EVM bytecode[7]. Intuitively, the contract can receive ether from other users, and its owner can send (part of) that ether to other users via the function `pay`. The hashtable `outflow` records all the addresses[8] to which it sends money, and associates to each of them the total transferred amount. The hashtable `inflow` records all the addresses from which it has received money.

All the ether received is held by the contract. Its amount is automatically recorded in `balance`: this is a special variable, which cannot be altered by the programmer. When a contract receives ether, it also executes a special function with no name, called *fallback*.

The function `AWallet` at line 6 is a constructor, run only once when the contract is created. The function `pay` sends `amount` *wei* ($1\,wei = 10^{-18} ETH$) from the contract to `recipient`. At line 9 the contract throws an exception

[7] Solidity is documented at solidity.readthedocs.io/en/develop/index.html

[8] Addresses are sequences of 160 bits which uniquely identify contracts and users.

if the caller (msg.sender) is not the owner, or if some ether (msg.value) is attached to the invocation and transferred to the contract. Since exceptions revert side effects, this ether is returned to the caller (who however loses the fee). At line 10, the call terminates if the required amount of ether is unavailable; in this case, there is no need to revert the state with an exception. At line 11, the contract updates the outflow registry, before transferring the ether to the recipient. The function send used at line 12 to this purpose presents some quirks, e.g. it may fail if the recipient is a contract. The fallback function at 16 is triggered upon receiving ether and no other function is invoked. In this case, the fallback function just updates the inflow registry. In both cases, when receiving ether and when sending, the total amount of ether of the contract, stored in variable this.balance, is automatically updated.

3 Collection of Ponzi schemes

In this section we describe our methodology for identifying Ponzi schemes and for extracting the related transactions.

3.1 Search of contracts with associated source code or web sites

We collect a set of Ethereum contracts from two sources:

- We use etherchain.org/contracts to retrieve Ethereum contracts which are associated with a name. In this way we find 86 contracts.
- We use etherscan.io to retrieve contracts with a verified source code. This means that the contract owner uploaded the source code (typically, written in Solidity) along with the contract address, and the repository verified that the result of the compilation matches the bytecode stored at the contract address. This search returned 1384 contracts.

After this preliminary collection phase, we manually inspect each of the above obtained contracts to identify Ponzi schemes. More specifically:

1. for each contract with a verified source code, we manually inspect its code (including comments) and, when available, the project web pages;
2. for each contract lacking a source code we perform targeted queries on Google and we seek out the official blog[9] and Reddit page[10] of Ethereum, looking for keywords like e.g. "Ponzi", "HYIP", "pyramid", "scam", "fraud", *etc.*, and examining the declared interest rates.

At the end of the this activity we end up with 137 contracts, which we make available online at goo.gl/CvdxBp (an excerpt is in Table 2). Notice that we had to exclude from our analysis some potential Ponzi schemes, like

[9] Ethereum blog: blog.ethereum.org

[10] Ethereum Reddit page: www.reddit.com/r/ethereum

e.g. ethtrade.org[11], ethereumlitemining.org, and other suspect scams listed at badbitcoin.org/thebadlist, because we did not manage to retrieve any information about the Ethereum addresses they use (if any).

3.2 Search of hidden Ponzi schemes

So far, our collection includes Ponzi schemes whose addresses have been advertised in some way, either by publishing them on a web site, or by associating them to the contract source code. However, other kinds of schemes may exist: those run by dedicated clients (which hide their addresses), those which have just been created for testing purposes, and those where the source code is either unavailable, or it has been published on external sites (and, therefore, is not certified). We perform a second search phase to include in our collection also some contracts of these kinds.

More specifically, we search the Ethereum blockchain for contracts whose bytecode (which is always stored on the blockchain) is *similar* to that of some known Ponzi scheme. We use the *normalized Levenshtein distance* [19] (NLD) as a measure of similarity between two bytecode files[12]. In order to establish when two contracts are similar, we first estimate the NLD between two *arbitrary* contracts on the blockchain. To this purpose we download the bytecode of *all* the contracts from etherscan.io, and we use a Monte Carlo algorithm to estimate the NLD between two random contracts. After these calculations, we estimate as 0.79 the NLD between two arbitrary bytecode files.

Then, we compute the NLD between the contracts in our initial sample, and *all* the contracts on etherscan.io. We classify as a Ponzi scheme any contract with a NLD less than 0.35 from some contract in our sample. The two values 0.35 and 0.79 are sufficiently far apart to ensure that, with very high probability, this method classifies as Ponzi schemes only the contracts that are highly similar to some contract in our initial sample. This search resulted in further 55 potential new Ponzi schemes, not included in our original collection of 137 contracts.

We then search in this set for false positives, i.e. contracts whose NLD from the initial sample is below 0.35, but they are not Ponzi schemes. To do that, we measure the NLD between the 55 contracts obtained in the previous step and (again) all the contracts on the Ethereum blockchain (but those in the original collection). We find only one match in this search: the contract with the shortest EVM code[13], which is close to thousands of contracts, and so we remove it from our collection. We conjecture that all the other 54 contracts in our collection of "hidden" schemes are Ponzi. Although we cannot support formally this claim (as

[11] Ethtrade Review — Is Ethtrade a Scam or Legitimate? Read Before You Invest. Source: managingyourfinance.com

[12] Roughly, the (non-normalized) Levenshtein distance between two strings measures the number of character which one has to change to transform the first string in the second one (e.g., the distance between "Ponzi" and "Banzai" is 3). The *normalized* version of the Levenshtein distance is a metric, and its value is a real number ranging between 0 (perfect equality) and 1 (perfect inequality).

[13] Address 0xada4347a112336c8a87bf91b85af275a22a41740, size of EVM code 1.2K.

no decompiler from EVM to Solidity is currently available), in practice we have observed that 0.35 is quite a discriminating threshold (recall that the average NLD between two arbitrary EVMs on the blockchain is 0.79). As an additional empirical support, we have observed that the Solidity code of Ponzi schemes is self-contained, i.e. it makes no use of external libraries. Therefore, it is quite unlikely that one of the contracts in our collection of 54 schemes is close to a Ponzi scheme just because they share a large portion of identical libraries.

For further validation, we have manually checked whether the "hidden" schemes are advertised in some forums as Ponzi schemes, if their Solidity source code is reported somewhere, or even if their functions have revealing names, like e.g.`awaitingPayout`, `nextPayoutGoal`, `changeFees`. The result is that, every time some additional information is found, it confirms the nature of the contract as a Ponzi scheme. Hence, we end up with a collection of 191 Ponzi schemes.

3.3 Extraction of transactions

For each contract identified in the previous steps, we gather all its transactions (both external and internal) from the Ethereum blockchain[14]. More specifically, for each transaction we record the following data: (i) the number of the enclosing block; (ii) the date when it was published on the blockchain; (iii) the address of the sender; (iv) the address of the receiver; (v) the amount of ether transferred by the transaction; (vi) a boolean value which records whether the transaction execution resulted in an error; (vii) a boolean value which indicates whether the transaction is external or internal.

To this purpose we have developed a suite of tools, which exploit the Parity client[15] and the `etherchain.io` API to extract transactions (both external and internal), and the Geth client[16] to associate them to timestamps.

4 Qualitative analysis of Ponzi schemes

In this section we analyse the source code of Ponzi schemes, to understand their behaviour, and find analogies between different schemes. We then discuss some security issues found in the analysed contracts. We also study how these schemes are promoted on the web, finding that in many cases there is a discrepancy between the advertisement and the actual chances of obtaining a payout.

4.1 Anatomy of Ponzi schemes

In order to understand how Ponzi schemes work in Ethereum, we manually inspect the source code of all the contracts identified in Section 3.1 that have

[14] Our dataset has been extracted on May 7th, 2017.

[15] Parity: `ethcore.io/parity.html`

[16] Geth: `github.com/ethereum/go-ethereum/wiki/geth`

an associated source code on etherscan.io[17]. We observe that all the contracts can be classified in one of the following categories:

- **array-based pyramid schemes** refund users in order of arrival. Generally, the schemes in this category promise to multiply the investment by a pre-specified factor. A user can redeem her multiplied investment when enough money is gathered from the users who later on join the scheme. We show in Figure 2 an archetypal array-based pyramid scheme which is very close, e.g., to Doubler1. To join the scheme, a user sends msg.amount ether to the contract, hence triggering the fallback function at line 14. The contract requires a minimum fee of 1 ether: if msg.amount is below this minimum, the user is rejected (line 15), otherwise, her address is inserted in the array (line 17), and the array length is incremented[18]. The contract sends 10% of the received amount to its owner (line 20), and with the remaining ether, it tries to pay back some previous users. If the balance is enough to pay the user in the array at position paying, then the contract sends to the user her investment multiplied by 2 (lines 23). After that, the contract tries to pay the next user in the array, and so on until the balance is enough. In this scheme, if a user gets some money, she knows exactly how much it is, and that amount is proportional to what she has invested. However, she must wait that all the users before her have redeemed their share.
- **tree-based pyramid schemes** use a tree data structure to record users's addresses. Each user has a parent, called *inviter*, except the root of the tree, which is the contract owner. Whenever a user joins the scheme, her money is split among its ancestors. We show in Figure 3 an archetypal scheme of this kind. To join the scheme, a user must send some money, and must indicate an inviter. If the amount is too low (line 15), or if the user is already present (line 16), or if the inviter does not exist (line 17), the user is rejected; otherwise she is inserted in the tree (line 19). After the user has been registered, her investment is shared among her ancestors (lines 25-29) halving the amount at each level. In this scheme, a user cannot foresee how much she will gain: this depends on how many users she is able to invite, and on how much they will invest. Unlike array-based schemes, her gain is not proportional to what she has invested.
- **handover schemes** store only the address of the last user: if someone wants to join, she must repay the last user of her investment plus a fixed interest. With this constraint, the amount that each user shall pay increases at each turn. An archetypal example is shown in Figure 4. To join the scheme a user must send at least price ether to the contract, hence triggering the fallback function of line 11. The contract forwards that sum to the former user, minus a fee which is kept within the contract (line 13). Then, the address of the new

[17] The examples we present assume version v0.2.2 of the Solidity compiler, which is the version used by most of the contracts in our collection. Although newer versions of Solidity change the way to declare functions and to manage arrays, these changes do not really affect the spirit of our examples.

[18] In Solidity, dynamic arrays can be resized by changing the length member.

user is recorded (line 14), and the `price` is increased of one half (line 15). The contract owner can withdraw his share by calling `sweepCommission`. In this scheme, if a user gets some money, she knows exactly how much it is, and the amount she can invest is fixed by the contract.

waterfall schemes divide each new investment among the already-joined users, starting from the first one. Each user receives a fixed percentage of what she has invested, as far as there is enough money. On the subsequent investment, the division starts again from the first user. We show in Figure 5 an archetypal scheme of this kind, which is very close, e.g., to `TreasureChest` and `PiggyBank`. To join the scheme, a user sends `msg.amount` ether to the contract, hence triggering the fallback function at line 18. The contract requires a minimum fee of 1 *ETH*: if `msg.amount` is below this minimum, the user is rejected (line 19), otherwise, her address is inserted in the array (line 21-22), and the array length is incremented. The contract sends 10% of the received ether to its owner (line 25), and with the remaining ether, it tries to pay back some previous users. If the `balance` is enough to pay the first user in the array, then the contract sends to that user 6% of her original investment (lines 29-30). After that, the contract tries to pay the next user in the array, and so on, until the balance is enough. On the next investment, the array will be iterated again, starting from the first user. In this scheme, the amount given to each user is proportional to what she has invested. However, it may happen that those late in the queue will never get any money at all, even when new users continue to join.

A common feature of these schemes, as we can see from their archetypal code (and remains true in the actual code), is that a user is never guaranteed to get back her investment.

Most of the contracts that we have analyzed belong to one of the categories described above. Almost the totality of them follow the array-based pyramid scheme. The tree-based scheme has been adopted by the two instances of the `Etheramid` contract. The handover scheme has been used in the various versions of `KingOfTheEtherThrone`; the waterfall scheme has been implemented by `TreasureChest` and `PiggyBank`.

By manually inspecting the source code of Ponzi schemes, we have observed very few differences among them, when none at all. Some contracts only differ in the multiplication factor, in the applied fees, or in the presence of auxiliary functions, like e.g. getter/setter for contract fields, or other utility functions for the owner. To have a precise estimate of the similarity between Ponzi schemes, we have computed the average normalized Levenshtein distance among their bytecode. We observe that this value is 0.54, which is far apart from 0.79, the average distance between the bytecode of two *arbitrary* contracts. This may suggest that most of the Ponzi scheme have been created by a single programmer, or that their code has been copied from the first instances appeared on `etherscan.io`.

```
contract ArrayPonzi {

  struct User {
    address addr;
    uint amount;
  }
  User[] public users;
  uint public paying = 0;
  address public owner;
  uint public totalUsers=0;
  function ArrayPonzi() {
    owner = msg.sender;
  }
  function() {
    if (msg.value < 1 ether) throw;

    users[users.length] = User({addr: msg.sender,
                                amount: msg.value});
    totalUsers += 1;
    owner.send(msg.value/10);

    while (this.balance > users[paying].amount * 2) {
      users[paying].addr.send(users[paying].amount * 2);
      paying += 1;
    }
  }
}
```

Fig. 2: An array-based pyramid scheme.

```
contract TreePonzi {

  struct User {
    address inviter;
    address itself;
  }
  mapping (address=>User) tree;
  address top;

  function TreePonzi() {
    tree[msg.sender] =
      User({itself: msg.sender,
           inviter:msg.sender});
    top = msg.sender;
  }
  function enter(address inviter) public {
    if ((msg.value < 1 ether) ||
      (tree[msg.sender].inviter != 0x0) ||
      (tree[inviter].inviter == 0x0)) throw;

    tree[msg.sender] = User({itself: msg.sender,
                             inviter: inviter});
    address current = inviter;
    uint amount = msg.value;
    while (next != top) {
      amount = amount/2;
      current.send(amount);
      current = tree[current].inviter;
    }
    current.send(amount);
  }}
```

Fig. 3: A tree-based pyramid scheme.

```
contract HandoverPonzi {
  address owner;
  address public user;
  uint public
    price = 100 finney;

  function HandoverPonzi() {
    owner = msg.sender;
    user = msg.sender;
  }
  function() {
    if (msg.value < price) throw;
    user.send(msg.value * 9 / 10);
    user = msg.address;
    price = price * 3 / 2;
  }

  function sweepCommission(uint amount)  {
    if (msg.sender == owner) owner.send(amount);
  }}
```

Fig. 4: An handover scheme.

```
contract WaterfallPonzi {

  struct User {
      address addr;
      uint amount;
  }

  User[] public users;

  uint pos = 0;
  uint public totalUsers=0;
  address public owner;
  uint public fees = 0;

  function WaterfallPonzi() {
    owner = msg.sender;
  }
  function() {
    if (msg.value < 1 ether) throw;

    users[totalUsers] = User({addr: msg.sender,
                            amount: msg.value});
    totalUsers += 1;
    fees = mgs.value / 10;
    owner.send(fees);

    pos=0;
    while (this.balance >= users[pos].amount *
        6/100 && pos<totalUsers){
      users[pos].etherAddress.send
          (users[pos].amount * 6/100);
      pos += 1;
}}}
```

Fig. 5: A waterfall scheme.

4.2 Security issues

Although one of the main selling points of "smart" Ponzi schemes is that the presence (and immutability) of source code makes them "reliable", our analysis has revealed several vulnerabilities, which undermine their trustworthiness. Some vulnerabilities are caused by poor programming skills, while some others are intentional: either should discourage a user to join such a scheme. However, to transmit a feeling of security, contract owners shelter themself behind the motto that *the code is publicly accessible* so eveyone can read it and decide whether or not to join the scheme. Since bugs are often missed even by their own creators, it is hard to imagine that the average user can read a contract and fully understand what it really does and what harms can be hidden behind.

Among the vulnerabilities caused by poor knowledge of the Solidity programming language, there is the bad management of the operation of paying user. As already noted in [2,8], if the `send` primitive (used in Solidity to transfer ether) fails, it returns an error code: if a contract does not check this error, it cannot acknowledge that there has been a problem. So, in case of errors during the `send`, the money remains within the contract, while the user will never get it. Notably, the large majority of the contracts we have analyzed do not check that the ether transfer succeeds. Their code is similar to the one in Figure 2 (line 19), Figure 3 (line 17), and Figure 4 (line 14). This vulnerability is known, at least, since the February 11st 2016, when the owner of `KingOfTheEtherThrone` realized that there was too much ether left on his contract[19].

Even when the return code of the `send` is checked, a careless managing can badly backfire, exposing the scheme to Denial-of-Service attacks and blackmailing. An example is given by scheme `HYIP` (see Figure 6), whose structure is similar to waterfall schemes. Here, investors are kept in an array, and they are all paid at the end of the day. The scheme checks that each `send` is successful: in case of errors, it throws an exception. Note that any error in one of the `send` (lines 25 and 31) will revert *all* the ether transfers. Errors may happen, for instance, for the following reasons: (i) the array of investors grows so long that its reading causes an out of gas exception; (ii) the balance of the contract finishes somehow in the middle of the `for` command (line 28), having not paid all the investors; (iii) one of the investor is a contract, whose fallback raises an exception. Indeed, exploiting the last scenario, an attacker could forge a contract with a fallback which always `throws` (see e.g., `Mallory` in Figure 6). The attacker contract sends a fraction of ether to `HYIP` to enter in the array of investors; when `HYIP` tries to send her the payout, the invoked fallback throws an exception. Note that there is no way to cancel Mallory from the investors array, hence `HYIP` is stuck and its balance frozen forever. At this point, the attacker could blackmail `HYIP`, asking for money to stop the attack (via the function `stopAttack`, line 21).

Although the unchecked `send` is the most widespread issue, there are other bugs which affect one or more contracts. For instance, `Government`[20], has a notorious bug, which has been found, so far, only in that contract. `Government`

[19] Source: www.reddit.com/r/ethereum/comments/44h1m1/

[20] `Government` is often called "GovernMental" or "PonziGovernMental" on web forums.

```
contract HYIP {
uint constant INTERVAL = 1 days;

struct Investor {
  address addr;
  uint amount;
}
Investor[] private investors;
address private owner;
uint private paidTime;

function HYIP() {
  owner = msg.sender;
  paidTime = now;
}

function() payable {
  investors.push(Investor(msg.sender, msg.
      value) );
}

function performPayouts() {
  if(paidTime + INTERVAL > now)  throw;

  uint fees = (this.balance * 37)/1000;
  if (!owner.send(fees)) throw;

  uint idx;
  for (idx = investors.length; idx-- > 0; ) {
    uint payout =
      (investors[idx].amount * 33) / 1000;
    if(!investors[idx].addr.send(payout))
      throw;
  }
  paidTime += INTERVAL;
}}
```

```
contract Mallory {

  address victim = 0x23...;
  address private owner;
  bool private attack = true;

  //to be created with
  //1wei of balance
  function Mallory() {
    owner = msg.sender;
  }

  function() payable {
    if (attack) throw;
  }

  function invest() {
    victim.send(1 wei);
  }

  function stopAttack(){
    if (msg.sender == owner)
      attack = false;
  }
}
```

Fig. 6: On the left, a snippet of the code of HYIP, a scheme vulnerable to Denial-of-Service attacks. On the right, the corresponding attack.

is an array-based Ponzi scheme with a quirk: in addition to the usual way to get back money if enough users keep investing, someone can win a jackpot if no one invests after him for 12 hours. The list of users is kept in an array, and when the 12 hours have expired, the array is cleared. However, the command used to clear the array had to scan *each* of its elements. At a certain point, the array grew so long that clearing every element required too much gas — more than the maximum allowed per single transaction. Hence, the contract got stuck, with the legit jackpot winner unable to claim her price.

Another bug concerns the constructor function, which is executed just once at creation time (usually, to initialize the owner of the contract with the address `msg.sender` of the sender of the first transaction). The constructor must have the same name of the contract, but we found four contracts where it has a wrong name: `GoodFellas`, `Rubixi`, `FirePonzi`, and `StackyGame`. Figure 7 shows an extract from the first two. On the left, `Goodfellas` has a function called `LittleCactus` (line 5) which sets the owner, and then the owner is sent the fees collected so far (line 11). On the right, `Rubixi` has a function called

```
1  contract Goodfellas {
2
3    address public owner;
4
5    function LittleCactus() {
6      owner = msg.sender;
7    }
8
9    function enter() {
10     ...
11     owner.send(collectedFees);
12     ...
13   }
```

```
1  contract Rubixi {
2    address private creator;
3
4    //Sets creator
5    function DynamicPyramid() {
6      creator = msg.sender;
7    }
8    //Fee functions for creator
9    function collectAllFees()  {
10     if (collectedFees == 0) throw;
11     creator.send(collectedFees);
12     collectedFees = 0;
13   }
```

Fig. 7: Constructor bug in `Goodfellas` and `Rubixi`.

`DynamicPyramid` (line 5) which sets the owner (called `creator`), and then there is a function `collectAllFees` which can be invoked to send the fees to the owner (line 11). The effect of giving the wrong name to a function which is meant to be a constructor is harmful: the function does not qualify to be a contructor at all, and it can be invoked by anyone at anytime, hence changing the owner address. We can see from the transaction list that when users discovered the bug, they started to invoke these functions to obtain the ownership and redeem the fees.

The contract `PiggyBank` contains two bugs which are extremely profitable for its owner[21]. According to the way `PiggyBank` is advertised[22], this contract is a waterfall scheme, where the owner keeps 3% fees, and each user receives 3% of their investment every time a new user joins the scheme. According to the advertisement, the command to compute the owner fees should be like:

```
fees = amount / 33
```

Actually, in the source code, the fees are computed as:

```
fees += amount / 33
```

The difference is subtle to spot, but relevant. With the second command, the fees grow at each deposit, and the consequence is that the owner share subtracted to each investment steadily increases. According to `etherscan.io`, the fees calculated for the seventh deposit exceeded the deposit itself.

The second bug of `PiggyBank` is related to the global variable used to scan the array, named `pos` in Figure 5. In `PiggyBank`, this variable is not reset, unlike in line 25. Hence, at each deposit, the iteration does not go from the *first* user to the last one, but from the *last* to the last itself. Hence, only one user at each deposit is paid, and only once. Notably, the conjunction of these two bugs resulted in giving (almost) all the money invested to the owner. Were only the second bug present, the contract would have kept accumulating a lot of unredeemable ether.

[21] Source: `www.reddit/piggybank_earn_eth_forever`

[22] Source: `bitcointalk.org/topic=1410587.0`

```
function init() private{
  //Ensures only tx with 1 ether
  if (msg.value < 1 ether) {
    collectedFees += msg.value;
    return;
  }...
```

```
function changeMultiplier(uint _mult){
  if (msg.sender != owner) throw;
  if (_mult > 300 || _mult < 120) throw;
    pyramidMultiplier = _mult;
}

function changeFeePercentage(uint _fee){
  if (msg.sender != owner) throw;
  if (_fee > 10) throw;
  feePercent = _fee;
}
```

Fig. 8: On the left, rejecting enrollment without returning the fee in Tomeka. On the right, the function used by the owner of TheGame to set multipliers and fees.

```
function Emergency() {
  if (owner!=msg.sender) throw;
  if (balance!=0){
    owner.send(balance);
    balance=0;
  }
}
```

```
function restart() {
  if (msg.sender==mainPlayer) {
    mainPlayer.send(address(this).balance);
    selfdestruct(mainPlayer);
  }
}
```

Fig. 9: On the left, withdrawing all the balance in EthVentures1. On the right, a termination function in TheGame.

Among the vulnerabilities caused by intentional programming choices, we have some that can harm the users. For instance, some contracts have a minimum fee to enter the scheme. If the fee is not met, the user is not allowed to join the scheme, and the sent amount should be returned. However, some contracts (e.g., DynamicPyramid, GreedPit, NanoPyramid, Tomeka), choose to keep the amount by themselves, without returning it to the user (see e.g. Figure 8 left). This is a questionable choice, especially when the minimum amount is quite relevant (e.g., in Tomeka the minimum is 1 *ETH*).

Another feature that could be used against users is the presence of functions which allow the owner to do special operations which can stray the contract from its expected behaviour. One example is DynamicPyramid, where the owner can change the interest rate, and also his fee shares (see Figure 8, right). A yet more harmful issue is present in those contracts (e.g., Doubler3, TheGame, ProtectTheCastle, DepositHolder, GreedPit, BestBankWithInterest, all the family of EthVentures) which allow the owner to withdraw all the money in the contract (see Figure 9, left), and in some cases, also to terminate the contract (see Figure 9, right). In both cases, withdrawing the accumulated money from the contract balance will have the effects of slowing down the process of repay users. Moreover, if the contract is also terminated, the users will have no chance to see their money again.

Note that all the array-based schemes can be easily shut down with a simple trick. To illustrate it, we consider the Doubler scheme, which sends back the amount multiplied by two. To perform the attack, Oscar needs to invest a large amount of ether (say, $100ETH$). Oscar first sends $100ETH$ to the contract, and then additional $100ETH$ (plus some fees)[23]. Upon receiving the second slot, the scheme will pay all the $200ETH$ back to Oscar, so he does not lose anything. From that moment on, all the subsequent investments will be gathered to pay back the second $100ETH$ of Oscar. If the average invested amount is smaller than $100ETH$, a large number of investors (and a lot of time) will be needed to pay back Oscar: hence, the scheme will not be able to pay out other investors for a while. Since the success of these schemes is based on the fact that they are fast to pay out, it is likely that with this attack, the scheme will be abandoned.

This attack can be performed at any time to disincentivize users to join a Ponzi scheme[24]. If performed at an early stage of the lifecycle of the Ponzi scheme, the attack succeeds with negligible money loss; otherwise, some money has to be given to repay previous investors.

4.3 Are "smart" Ponzi schemes frauds or social games?

Many creators of Ponzi schemes promote them as mere "social games", in contrast with the fact that Ponzi schemes are illegal in many countries (actually, Ethereum lies in a grey zone of jurisdiction, where it is unclear which laws apply). By analysing the websites of many of the Ponzi schemes in our collection, we took the view that the way they are advertised is fraudulent, since users are mislead about the actual probability they have of getting their investment back [6]. In many cases, the possibility of receiving a payout is presented as extremely likely, while we will see later on that it is not so.

For instance, the array-based Ponzi scheme `DianaEthereum-x1.8` is introduced as follows[25]:

> "*Hello! My name is Diana. I attract good luck.[...] You should send me ether and you'll get your money multiplied by 1.8.*"

With these words, it seems that the money will be sent back *for sure* to everyone. However, as shown in Table 2, only 83 users out of 125 were paid back, for a total of 63 100 *USD*.

Some schemes are presented as "high-yield" investment programs. For instance, `EthStick` is advertised as follows[26]:

> "*The mechanics are quite simple: deposit up to 5ETH and get a 20% return when enough people deposit after you (the settings can be changed to adapt to the trends, but only within defined limits).[...] I believe it's a decent investment opportunity, and I'm reinvesting part of my earnings myself, as you can see on the ranking.* "

[23] To guarantee the atomicity of the sends, Oscar will use a contract to send the money.

[24] As far as we know, this attack has been performed only on contract `Quadrupler`. See etherscan and bitcointalk for details.

[25] Source: `bitcointalk.org/index.php?topic=1402534.0`

[26] Source: `reddit.com/etherstick`

Here, the owner is fair admitting that *settings can be changed*, but actually he means that he can change (raise) the fees he is getting, and also that he can change (lower) the multiplier factor which calculates how much is to be given back to each user. Here, users joining the scheme have to trust him to behave correctly, hence failing the main purpose of a smart contract — not having to trust any third party.

As another example, `TheSimpleGame` is presented as follows[27]:

> "*Automatically you get what you invested +25% additional ETH. [...] You get your coins after other people deposit, so the more people are playing the game, the faster you are getting your coins.*"

Here, it seems that the only drawback is the possibility for the game to *slow down*, while the possibility of never giving the money back is not even mentioned.

To give a semblance of trustworthiness, the contract `GreedPit` is introduced as follows[28]:

> "*Tired of copy paste Ponzi games? Me too. This one has gone through weeks of brainstorming and tweaking to provide a solid, enjoyable (and hopefully profitable) experience.*"

However, this one too is an array-based Ponzi scheme, which has paid out only 13 users of 61, receiving 10 418 *USD* (see `goo.gl/CvdxBp`).

As a last example of unfulfilled promises, we show the advertising of `Rubixi`[29], the contract with the constructor bug highlighted in Section 4.2:

> "*Hello! My name is Rubixi! I am a new and verified pyramid smart contract running on the Ethereum blockchain. When you send me 1ETH, I will multiply the amount and send it back to your address when the balance is sufficient. My multiplier factor is dynamic thus my payouts are accelerated and guaranteed for months to come.*"

Rubixi promised to last long and to give lots of payout; however, upon a creation date on 14th of March 2016, the last deposit was done on the 9th of April 2016. Then, the bug was discovered, and no one has sent money any longer. Now, the contract has a balance of $4 ETH$, (being worth $360.29 USD$ on the 15th of May 2017) and it has paid out only 23 users of 98 (see Table 2).

Excluding the unfortunate (but extremely likely) case of a bug which causes the users to lose interest and trust the scheme, we see from `goo.gl/CvdxBp` that the trend is similar for all the contracts in our collection: the amount of users actually paid out is far smaller than what one might expect according to the advertisement. For instance, `DynamicPyramid`, the most successful in term of money raised, has paid only 51 users out of 174. `EthereumPyramid`, the first contract by number of users, has paid only 124 out of 326; `ZeroPonzi`, a scheme which subtracts no fees from the invested money, has paid 28 users out of 46.

[27] Source: `bitcointalk.org/index.php?topic=1424959.0`

[28] Source: `bitcointalk.org/index.php?topic=1415365.0`

[29] Source: `bitcointalk.org/index.php?topic=1400536.0`

This behaviour is not surprising. To clarify it, consider a simple array-based scheme which doubles the received money, and accepts entry tolls of exactly $1ETH$. In this scheme, there are no fees: the owner just gets the first $1ETH$ sent to the contract. Assume that the first user U_1 sends $1ETH$. His money is given to the owner, and so it is removed from the contract. The balance is 0. For U_1 to see back his $1ETH$ plus the other one promised, he must wait for two others users U_2 and U_3 to join the scheme, by sending $1ETH$ each.

In Figure 10, each node represents one user, and its children are the users needed to redeem his share. So, U_2 must wait for U_1 to redeem his share, and then he must wait for U_3 and U_4 to send money (hence he has to wait a total of 3 users). User U_3, who is the last one on his level, must wait that all the subsequent level is full, which gives a total of 4 users to wait. In general, a user U_k at level i must wait that all those users on the previous level have redeemed their share (e.g., U_6 and U_7), and then he must wait for all the ones on his level that have arrived before him. If U_k is the first node at level i, he must wait for all the other users at level i to join, plus the two ones needed to redeem his share. This needs $2^i - 1 + 2$ users. Since the amount of nodes up to level $i - 1$ is $2^i - 1$ and since U_k is the first at level i, we can say that $k = 2^i$ and hence, in the best case, U_k must wait $k + 1$ users. Instead, if U_k is the last user at level i, he must wait for all the other users at level $i + 1$. This needs 2^{i+1} users. Since, $k = 2^{i+1} - 1$, in this case U_k must wait for $k + 1$ users. For instance, a user joining this scheme in position, say, 50th will have to wait exactly 51 other users to join. In general, the first ones to join have better chances to see their payout.

Although this simple example considers a scheme with no fees and a fixed amount of money from each user, the general considerations about the chances of redeeming one's investment remain true for all the contracts in our collection.

In the contracts which pose no limit on how much one can invest, an unusually high investment could make the contract stop sending payouts for a lot of time, while accumulating the payout, thus discouraging new users to join. This is the case, e.g., of `Doubler2`, an array-based scheme which doubles the invested amount, and has 10% fee and a minimum entry amount of $1ETH$. However, the contract has paid out only up to the 68th user out of 210. Looking at the list of its transactions, we see that the most common toll is of $1ETH$-$5ETH$ but here and there, there are some higher ones (up to $50ETH$) which make the system very slow to fill up a level. In general, the higher the promised profit (i.e., `MultiplyX10`), the slower the scheme is.

We now analyze `PiggyBank` and `TreasureChest`, the two waterfall schemes in our collection. As we have seen in Section 4.2, `PiggyBank` was flawed, and hence it was played only for a couple of days. However, this is a lot of time, considering that no user received the promised payout, even not the first ones to join the scheme. Probably, the users were fooled by the owner, who declared his skill in programming and testing, and in the fact that, in other Ponzi schemes, one had to wait a bit to see the first payouts[30]:

[30] Source: bitcointalk.org/index.php?topic=1410587.80

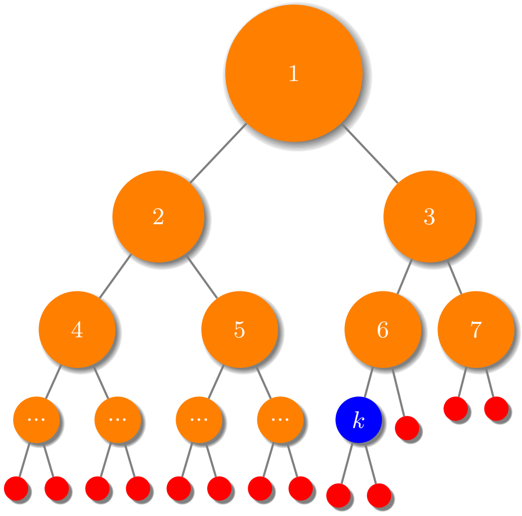

Fig. 10: Payout tree for a scheme which doubles the invested money and accepts exactly 1ETH from each user. The first ether is given to the owner.

> "*[...] A lot of other games are flawed [...] This is not the case with PIGGY-BANK, I have tested the code rigurously, and it works perfectly, the balance will be paid out after a while.*"

Having discovered the bugs, someone created an improved version of it, called `TreasureChest`. This contract was advertised as follows[31]

> "*TreasureChest: Earn from Ethereum forever – TreasureChest is a new game [...] where you can earn a stable 6% profit instantly and forever.[...] You get paid every single time a new investor joins or invests again. After 5 investor, you will earn 30% profit. After 50 investor, you will earn 300% profit. [...]* "

However, having fixed the bugs did not make the promises more reliable. Indeed, unlike array-based and tree-based schemes, in a waterfall scheme only the first users in the array earn money at each new investment. Consider e.g. the case where each user sends 1ETH, and there are no fees for the owner. Then, according to the advertisement, each users should receive 0.06ETH each time someone joins, but this amount can be sent only to 16 users, since after that there is no money left. Hence, in this situation, only the first 16 users would profit from new deposits. Also, assuming that one is among the first 16 users, then she must wait at least 16 other deposits before having, at least, repaid her initial investment. In `TreasureChest` the amount for the initial deposit was not fixed, so at each investment the index i such that the i-th user was paid changed. This might have made some of the later users believe that the advertised promises were somehow trustworthy.

[31] Sources: `bitcointalk.org/index.php?topic=1413721.msg14326777#msg14326777` and `www.docdroid.net/treasurechest.pdf.html`

5 Impact of Ponzi schemes

In Table 1 we draw some general statistics about all the 191 schemes obtained from our collection phase. Full data about the collected Ponzi schemes, including their unique addresses, are reported online at `goo.gl/CvdxBp`. Table 2 shows the first 10 contracts in our list, ordered by total amount of invested ether.

Table 1: Statistics about Ponzi schemes on Ethereum.

		#Trans.		*ETH*		*USD*		Users	
Kind	# Schemes	in	out	in	out	in	out	paying	paid
Public	137	12773	7995	42 367	41 861	411 919	412 903	2103	1076
Hidden	54	5004	305	1189	1029	6842	5920	201	123
Total	191	17777	8300	43 556	42 890	418 761	418 823	2304	1199

The leftmost column in Table 1 contains the kind of contracts: *public* are those collected through the methodology in Section 3.1, while *hidden* contracts have been collected using the technique described in Section 3.2. The second column contains the number of schemes of the given kind. The other columns report the number of incoming and outcoming transactions, and the overall transferred value, both in *ETH* and in *USD* (rounded to an integer). To convert the amount of each transaction to *USD*, we use the average exchange rate on the day of the transaction, obtained from `etherchain.org`[32]. Note that the value transferred through a transaction has a different meaning, according to whether the transaction is external or internal:

- external transactions are created by users, to invoke contract functions. These transactions can transfer some ether from a user to the called contract. Hence, this amount of ether is part of the "incoming" value of the contract.
- internal transactions are triggered by the execution of some external transactions. The actual meaning of an an internal transaction related to a Ponzi scheme depends on its fields "from" an "to". If the "from" address is the one of the contract under observation, then the value transferred by the transaction constitutes the outgoing payout from the scheme (possibly, part of this payout goes to contract owner). Instead, if the "from" address is that of another contract, then the transaction sends money from that contract to the one under observation (which is referred in the "to" field of the transaction). This may happen because, instead of sending her money directly to a Ponzi scheme, a user exploits another contract (typically, a wallet contract).

Note that, similarly to [17] for Ponzi schemes on Bitcoin, also in Ethereum we cannot precisely quantify the profit of scammers, since we do not know how to

[32] Source: `etherchain.org/api/statistics/price`

Table 2: Top-10 Ponzi schemes by amount of invested ether.

	#Trans.		*ETH*		*USD*		Users		Transactions	
Contract name	in	out	in	out	in	out	paying	paid	first	last
`DynamicPyramid`	418	143	7474	7437	83230	82704	174	51	2016-02-23	2016-11-12
`DianaEthereum-x1.8`	277	166	5307	5303	63100	63100	125	83	2016-03-08	2017-04-09
`Doubler2`	380	161	4858	4825	25432	25306	210	68	2016-02-16	2016-08-10
`ZeroPonzi`	626	499	4490	4489	50867	50857	46	28	2016-04-04	2016-04-06
`Doubler`	151	53	2977	2950	14251	13971	91	16	2016-02-19	2017-04-12
`Government`	723	846	2938	2938	36311	42396	40	27	2016-03-08	2017-03-20
`Rubixi`	624	61	1367	1363	16715	16593	98	23	2016-03-14	2016-10-14
`ProtectTheCastle1`	765	757	1144	1138	12737	12761	98	65	2016-03-20	2017-01-29
`EthereumPyramid`	965	338	986	917	4929	5178	326	124	2015-09-07	2016-08-28

separate, in internal transactions, the money sent to legit users from the money sent to scammers. A rough over-approximation of the profit of scammers is the value exchanged through external transactions.

The columns "Paying users" and "Paid users" in Table 1 indicate, respectively, the number of users entered in the scheme (i.e., the distinct addresses that send money to the contract), and the number of users that have subsequently received a payment from the contract.

Measuring the impact of Ponzi schemes. The data in Table 1 give a first measure of the impact of Ponzi schemes on Ethereum. First, we compare the overall number of transactions related to Ponzi schemes against the transactions in the whole Ethereum blockchain. To this purpose, we count the number of transactions from July 30, 2015 (the date of the origin block in Ethereum) to May 7th, 2017 (the date when we extracted the transactions), obtaining a total of 16082269 transactions. Since we counted 17777 transactions related to Ponzi schemes, we have that Ponzi schemes only constitute $\sim 0.05\%$ of the transactions in the Ethereum blockchain.

We measure the economic impact of Ponzi schemes, by quantifying the overall value in USD exchanged through them. We choose USD, rather then ETH, as a unit of measure because the exchange rate of ETH has been highly volatile, ranging from a minimum of $\sim 0.5\,USD$ in October 2015 to a maximum of $\sim 100\,USD$ in May 2017 (see Figure 11). Overall, we observe that the Ponzi schemes in our list collected 418 761 USD from 2304 distinct users.

Note that the difference between incoming and outgoing ETH is always non-negative, because contracts cannot send more ETH than what they receive. Instead, the difference between incoming and outgoing USD can be negative. This is not a contradiction: it can be explained by the fact that the exchange rate between ETH and USD has varied over time, as depicted in Figure 11. For instance, $1\,USD$ deposited to a contract in December 2015 (when the exchange was $1\,USD \sim 1\,ETH$) and withdrawn in May 2017, resulted in a gain of $\sim 100\,USD$.

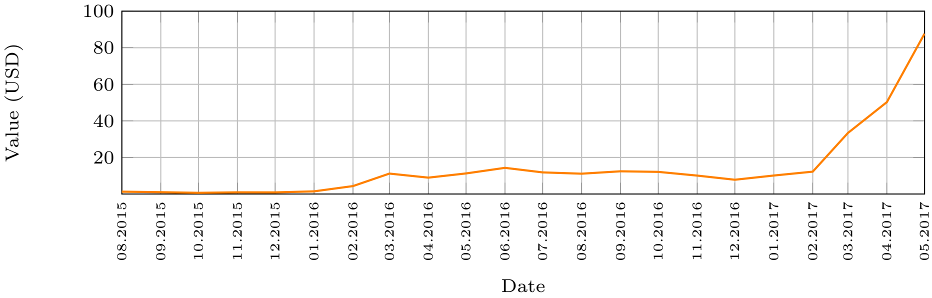

Fig. 11: Ether/USD exchange rate.

6 Measuring the scheme creation and lifetime

We now study the lifetime, measured as the number of days from the first to the last transaction, of all the Ponzi schemes in our collection. Figure 12 underlines the short lifetime that characterizes Ponzi schemes (the average is 104 days). The solid line also shows that $\sim 75\%$ of public Ponzi schemes have a lifetime of 0 days. Basically, this means that they were deployed on the Ethereum blockchain, and in many cases advertised by forums or dedicated web sites, but they did not manage to attract any users. The short lifetime reflects the typical behavior of pyramidal schemes, characterized by an high number of transactions operated in a short time frame (usually between the beginning and the middle part of its lifetime), before the scheme is abandoned.

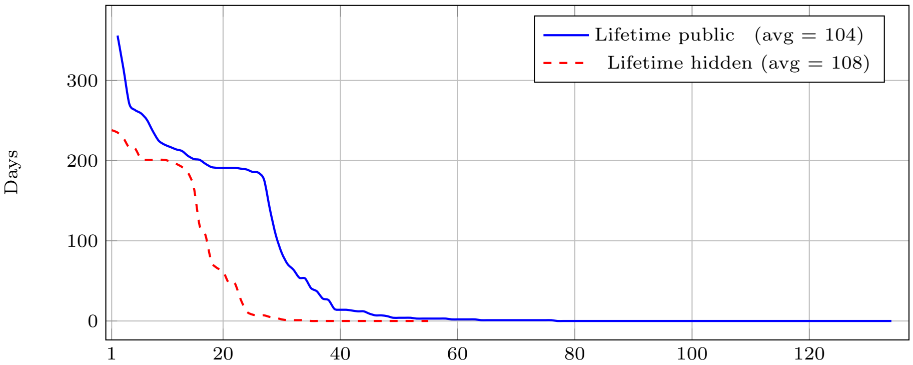

Fig. 12: Lifetime of Ponzi schemes. On the x-axis, the number of contracts; on the y-axis, their lifetime measured in days.

Figure 13 shows how many Ponzi schemes have been created over time. We see a peak in April 2016, with 87 new public Ponzi schemes, and 71 new hidden schemes. After this first wave of creations, the situation has settled, with an average of ∼ 3 new public schemes per month. We conjecture that the fall in the number of creations of Ponzi schemes is somehow related to the attack to the DAO, a contract implementing a crowd-funding platform, which raised $\sim \$150M$ before being attacked on June 18th, 2016[33]. An attacker, who exploited a bug in the DAO contract[34] managed to put $\sim \$60M$ under her control, before the hard-fork of the blockchain nullified the effects of the transactions involved in the attack. This event had several side effects on Ethereum, making its capitalization fall in the days following the attack, and in general discrediting its reputation of trustworthy platform[35]. We believe that the second wave of Ponzi schemes on Ethereum will be harder to identify, as scammers will devise less explicit ways of attracting and cheating their victims. We discuss in Section 10 some new potential Ethereum-based frauds.

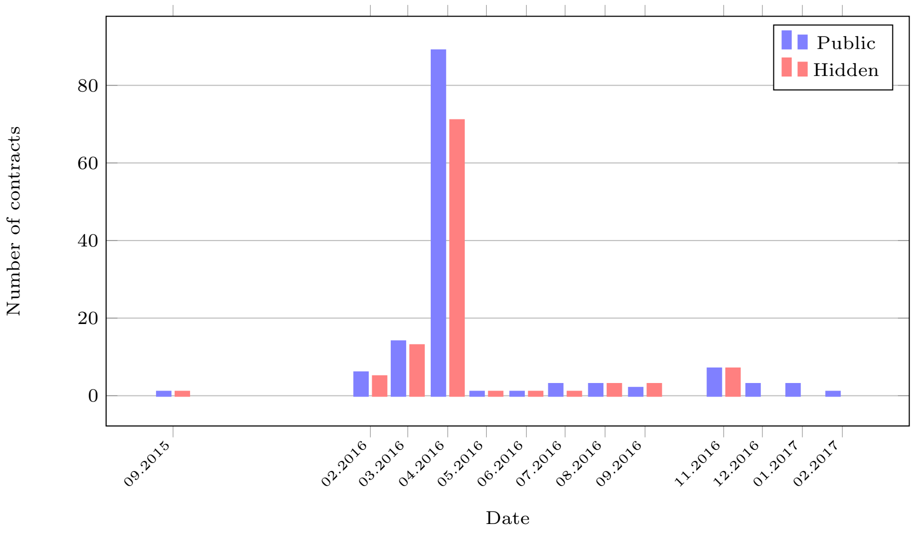

Fig. 13: Creation of Ponzi schemes.

7 Measuring gains and losses

We now study the distribution of gains and losses among users. We expect to observe the common pyramidal pattern of Ponzi schemes, where only a few users earn money, while the vast majority of users lose their investment. This kind of

[33] www.coindesk.com/understanding-dao-hack-journalists/

[34] hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit

[35] https://bitcoinmagazine.com/articles/ethereum-s-dao-forking-crisis-the-bitcoin-perspective-1467404395/

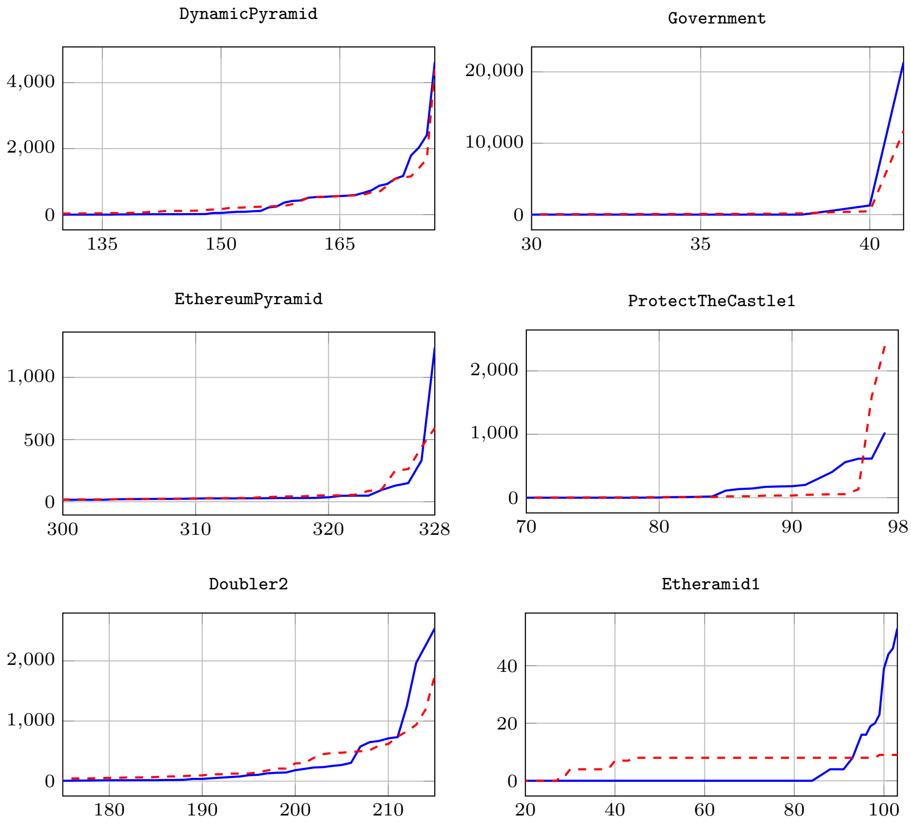

Fig. 14: Gains and Losses by users. On the x-axis, the number of users; on the y-axis, the *USD* gained (blue solid line) and lost (red dashed line) by each user.

observation requires to study the pattern of gains and losses individually for each contract. To save space, we restrict our analysis to a small sample of contracts.

Figure 14 shows the gains and losses of users. On the x-axis we have a point for each user involved in the scheme; the y-axis represents the gains (solid blue line) and losses (dashed red line), measured in *USD* and sorted in ascending order. In some diagrams we prune the left part of the curves, to highlight the characteristic behaviour of Ponzi schemes shown in their tail. In all diagrams, the left part shows a majority of users whose gains and losses are close to zero. For instance, the curves in the diagram for `EthereumPyramid` show that ~ 300 users have neither gained nor lost almost anything; the solid line shows, in particular, shows that one user has gained more than 1000 *USD* (we suspect it to be the contract owner); the dashed line shows that 8 users have lost up to ~ 600 *USD*.

We can observe that all the diagrams in Figure 14 share a similar pattern: as the payoff increases, the number of users who gain that payoff decreases; in any case, the users who lose money are more than those who gain.

8 Measuring the volume of payments

In this section we study how Ponzi schemes perform over time. Figure 15 shows the daily volume of payments (measured in *USD*) of all the 191 Ponzi schemes in our collection. The x-axis represents time, and the y-axis gives the volume of money transferred (measured in *USD*). The red dashed line represents money sent by users to the schemes, while the blue solid line represents money sent by the schemes to users. The diagram clearly reports an equilibrium between outcoming and incoming flows, meaning that most of the money invested in the schemes are redistributed to users. However, the distribution of money follows the pattern of inequality that characterizes Ponzi schemes, as highlighted in Section 7, and further discussed later on in Section 9.

From Figure 15 we observe that most value was exchanged in the period from February to May 2016, with three peaks between March and April 2016. It is plausible that the fall of activity after April 2016 is a consequence of the analogous drop in the creation of new Ponzi schemes, witnessed in Figure 13.

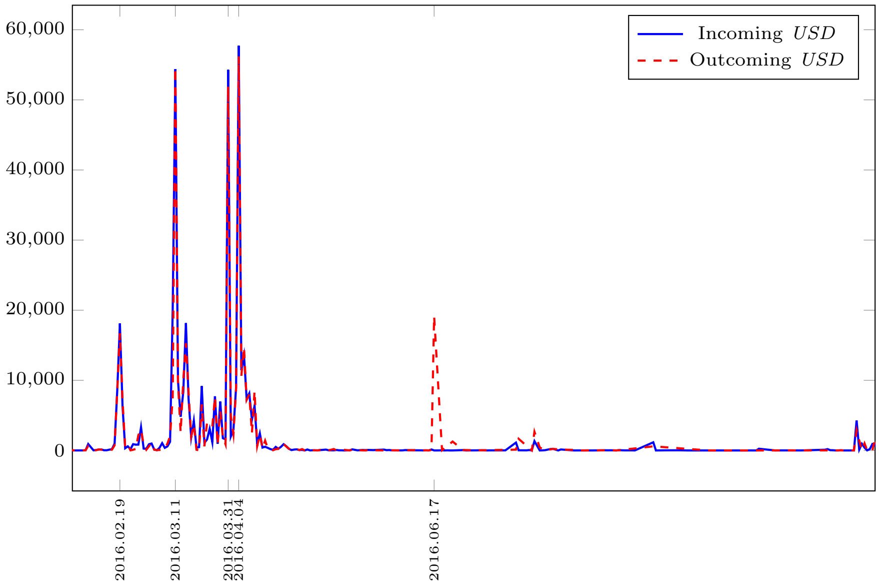

Fig. 15: Daily volume of transactions (complete set of 191 Ponzi schemes).

We now measure the volume of transactions pointwise, on a sample of the most representative schemes. Each diagram in Figure 16 shows the money flow (in and out) of a single contract: the red dashed lines represent money sent to the scheme (measured in *USD*), while the blue solid lines represent payouts sent by the scheme to users. The x-axis represents time: we consider the total incoming/outgoing money per day.

In the diagram for DynamicPyramid, we see that the most of the incoming flow happened on the 11st of March. Indeed, according to the list of transactions on etherscan.io, the total investments in that day alone amount to almost 60 000 *USD*. We see that the blue and red flows in that day almost overlap, meaning that with such a great balance the contract was able to pay out many of users. However, after that single peculiar day, users almost stopped sending ether, and so did the contract. Now the contract has a balance of $37ETH$, and it is waiting to pay out the 52nd user of 174.

The diagram of Government is peculiar, due to a bug which affected it (already discussed in Section 4.2). Periodically, this contract needs to clear the array which records the list of users that joined the scheme: however, from a certain point performing this operation would have required more gas than the maximum allowed for a single transaction. Consequently, several attempts to clear the array and to redeem the jackpot have failed with an "out-of-gas" exception. Exactly in the date of the first hard-fork (on the 17th of June), which also raised the gas limit[36], we observe an internal transaction of 22 699 *USD*, used to withdraw the jackpot and correctly clear the array.

The diagram for EthereumPyramid shows that the investments were made basically in two slots of time: one around the last days of February and another on a single day, the 1st of April. From etherscan.io we see that all those later investments were strangely made by the owner: they were almost 50 in a single day. We see that the inflow and outflow almost overlap. EthereumPyramid asks to all users exactly $1ETH$ and triples the invested money. With such a fixed toll, every three users one is paid out, and the outflows is smooth. However, we see that there is a peak in the outflow around the 26th of June. That day, a single payment has been made of $90ETH$. After inspecting the code and the set of transactions, we are inclined to say that it is the owner withdrawing her fees.

From the diagram of Etheramid we see that there is perfect overlap of inflow and outflow: recall that it is a tree-based Ponzi, and that everything which goes in is immediately sent to the users's ancestors. There is no need to delay payments waiting for the payout to reach its quote, like in array-based schemes (for instance, see the diagram of Doubler2).

9 Measuring payment inequality

Our last analysis measures the inequality in the distribution of investments along the schemes in our sample. To this purpose we use *Lorenz curves* (Figures 17 and 18) and *Gini coefficients* (Figure 19), two standard graphical representations of the distribution of income or wealth.

The Lorenz curves represents users on the x-axis (in percentage), and on the y-axis the percentage of payments *into* (Figure 17) and *from* (Figure 18) the Ponzi scheme. A diagonal line at 45 degrees from the two extremes of the diagram (leftmost-bottomost to rightmost-topmost) represents the perfect equality: i.e.,

[36] blog.ethereum.org/2016/07/20/hard-fork-completed/

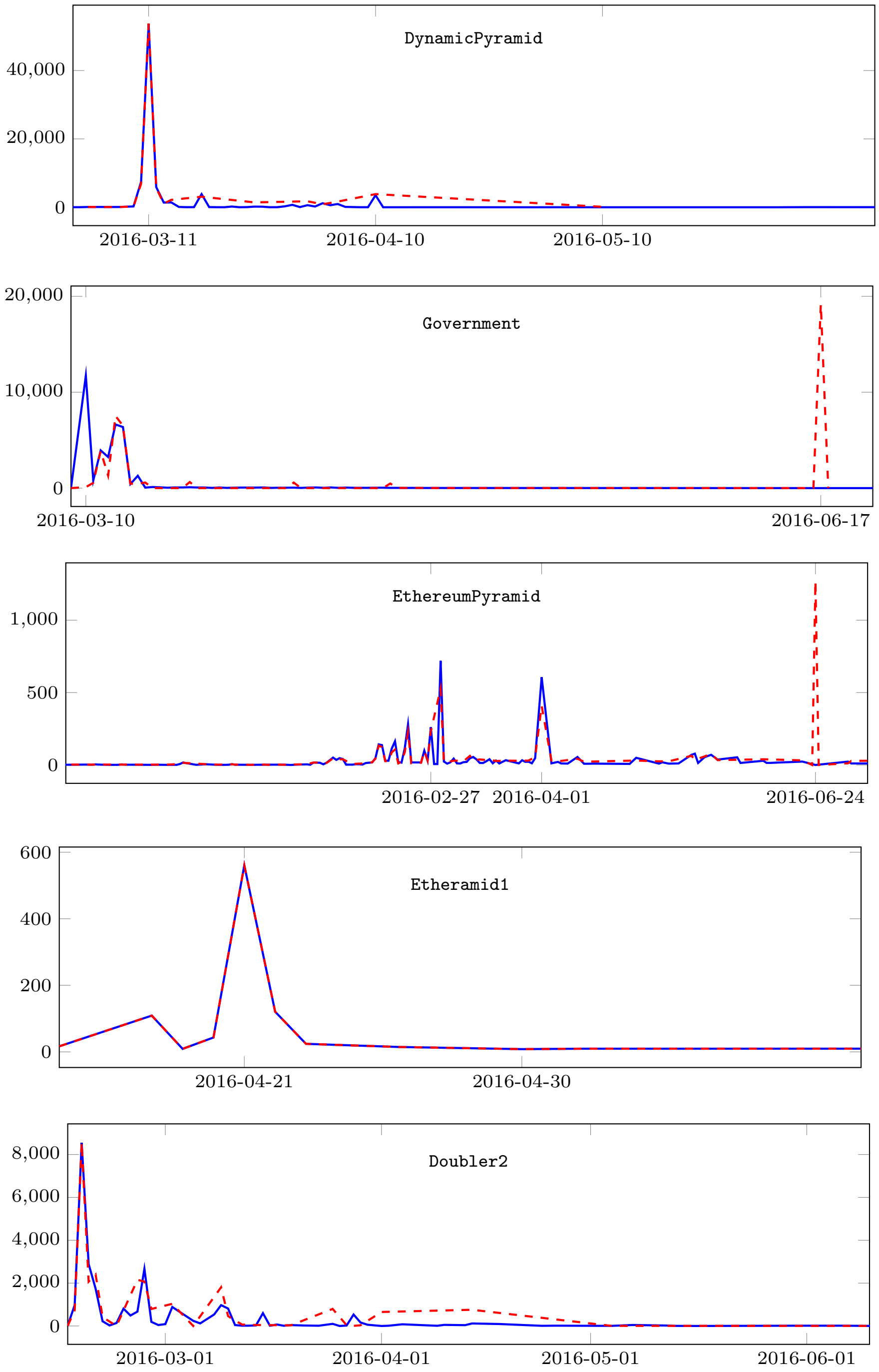

Fig. 16: Volume of payments into and out Ponzi schemes, by time. On the x-axis, the dates of transactions; on the y-axis, the *USD* sent to (blue solid line) and from (red dashed line) the contract.

for all $x \in [0, 100]$, the x% of the whole population of users has invested/received the x% of the total income of the scheme. Instead, the perfect disequality is represented by the (discontinuous) function that has value 0 for all $x < 100$, and value 100 for $x = 100$: this means that a single user has invested/received the total sum in the scheme.

We can observe in Figure 17 that `Etheramid1` is quite close to perfect equality, while the most unbalanced schemes in our sample are `Government` and `ProtectTheCastle`, where 10% of victims have invested more than 90% of the money. The Lorenz curves of these two schemes are quite close to the overall curve of Bitcoin-only Ponzi schemes in [16]. Overall, the closer is a curve to the one which represents perfect inequality, the more a Ponzi scheme benefits from "big fishes" who invest large amounts of money in the scheme; dually, if the curve is close to the one which represents perfect equality, the scheme benefits from a large population of victims who invest a small amount of money.

From Figure 18 we observe that the distribution of payouts is in general more iniquitous than that of investments, as the Lorenz curves are more squeezed to the right, compared to those in Figure 17. Interestingly enough, although `Etheramid1` is almost perfectly balanced for investments, when we see the distribution of payouts we see that it is quite unbalanced.

The Gini coefficients in Figure 19 relate the inequality of investments/payouts to the "success" of the scheme, defined as total amount of money invested/received by users. The x-axis represents the degree of inequality (0 indicates perfect equality, while 100 is perfect inequality), and the y-axis measures the total investment/payout. Each scheme is represented by an arrow, whose tail represents investments, while the head represents payouts. For the most lucrative scheme, `DynamicPyramid`, we observe that the index of inequality is high, surpassing 80% for both investments and payouts. For `ProtectTheCastle`, we see that the head and the tail of the arrow almost overlap, meaning that the inequality distributions of investments and payouts are very close in this scheme. For the less lucrative schemes, no correlation seems to exist bewteen the success of the scheme and the index of inequality.

10 Conclusions

Blockchains and smart contracts might really be the next "disruptive" technology, as many companies, newspapers, and researchers start to believe. However, they can also offer new opportunities to tax-evaders, criminals, and fraudsters [4, 14], who can take advantage of their anonymity and decentralization. In this survey we have analysed the impact of Ponzi schemes on Ethereum, the most flexible and widespread platform for smart contracts so far, with a market capitalization that has reached 8 billion *USD*[37].

Overall, we have observed that, in these first 2 years of life of Ethereum, there have been a multitude of experiments to implement Ponzi schemes as

[37] coinmarketcap.com/currencies/ethereum

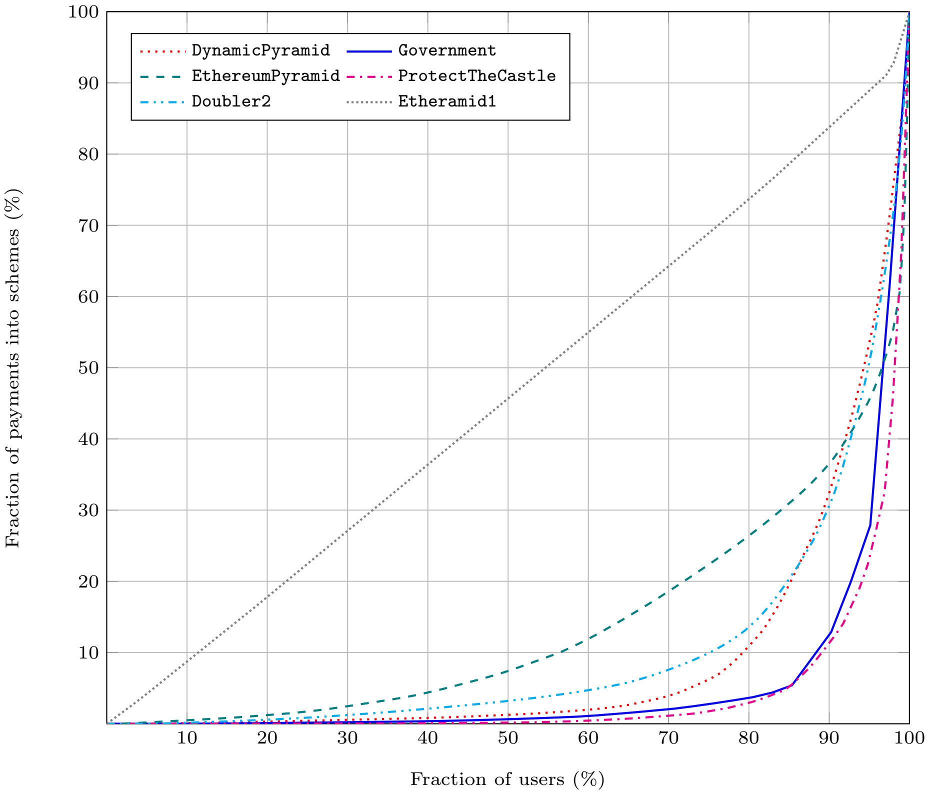

Fig. 17: Lorenz curves of a sample of Ponzi schemes (payments in).

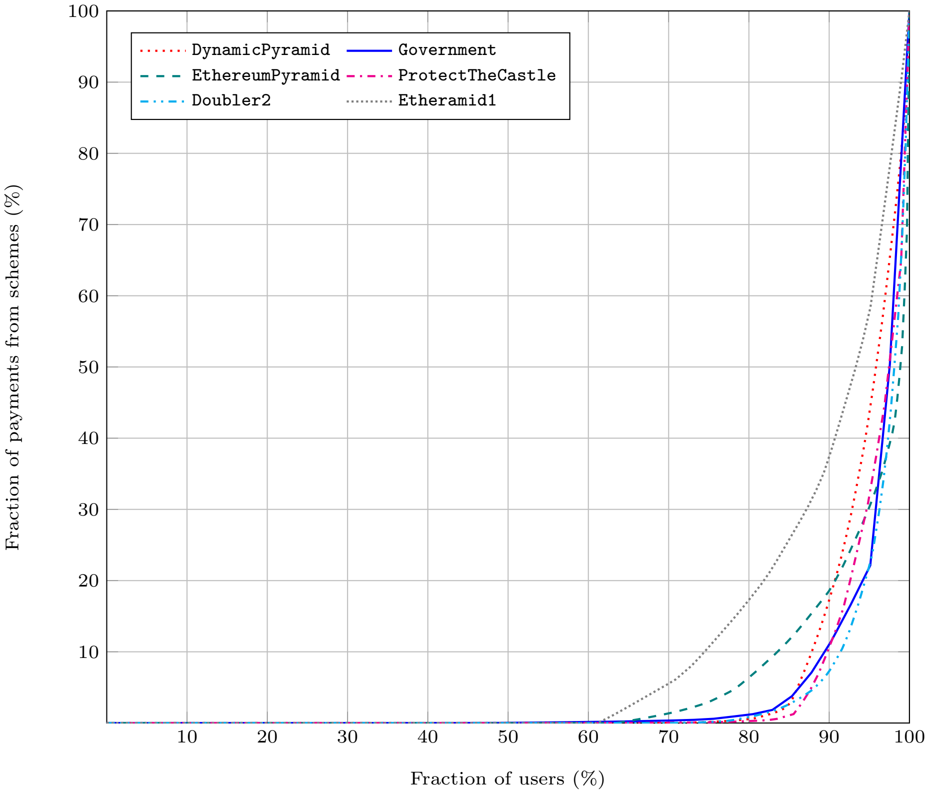

Fig. 18: Lorenz curves of a sample of Ponzi schemes (payments out).

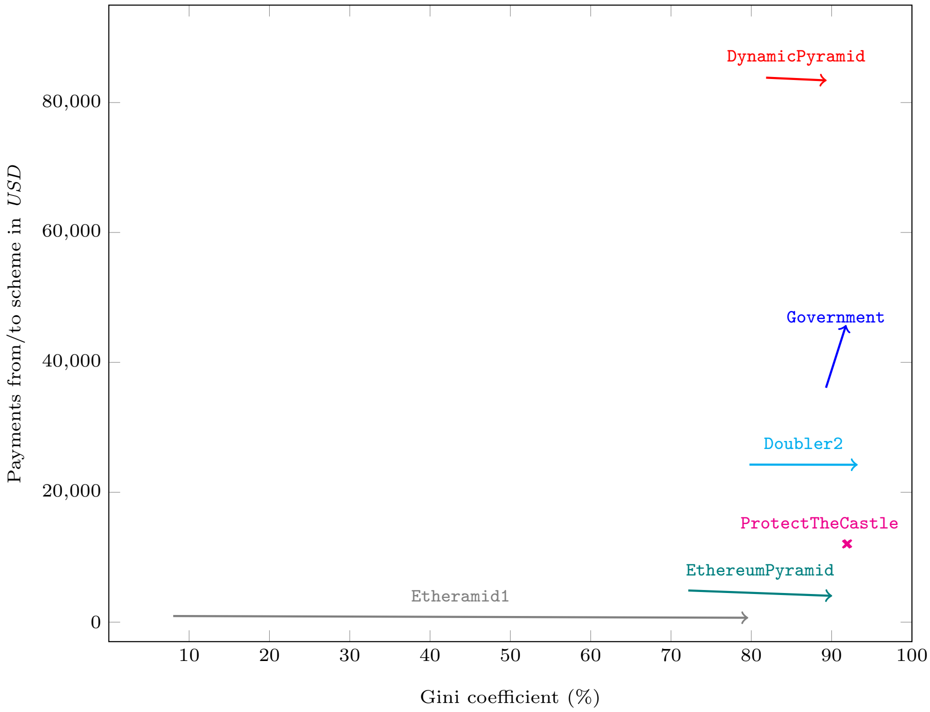

Fig. 19: Gini coefficients of a sample of Ponzi schemes.

smart contracts: indeed, $\sim 10\%$ of the 1384 contracts with verified source code on etherscan.io are Ponzi schemes. However, the impact of these experiments is still limited, as only $\sim 0.05\%$ of the transactions on the Ethereum blockchain are related to Ponzi schemes.

Still, it is foreseeable that, as Ethereum consolidates its position as a platform for smart contracts and as a cryptocurrency, criminals will exploit it to host their scams. Besides the growth of traditional Ponzi schemes accepting ether[38], we expect a second wave of Ponzi schemes, but very likely they will be less recognizable as such than the ones collected in this survey. For instance, they could mix multi-level marketing, token sales, and games, to realize complex smart contracts, which would be very hard to correctly classify as Ponzi schemes or legit investments[39].

Our analysis suggests that there is still time to devise suitable policy interventions, and provides a first understanding of the issues that must be tackled in this direction.

[38] badbitcoin.org/thebadlist
[39] https://steemit.com/crypto-news/@forklognews/forklog-users-accuse-blockchain-based-lottery-kibo-of-scam

Acknowledgments. This work is partially supported by Aut. Reg. of Sardinia project P.I.A. 2013 “NOMAD”.

References

1. Artzrouni, M.: The mathematics of Ponzi schemes. Mathematical Social Sciences 58(2), 190–201 (2009), http://dx.doi.org/10.1016/j.mathsocsci.2009.05.003
2. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts (SoK). In: Principles of Security and Trust (POST). LNCS, vol. 10204, pp. 164–186. Springer (2017), http://dx.doi.org/10.1007/978-3-662-54455-6_8
3. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In: IEEE S & P. pp. 104–121 (2015)
4. Brito, J., Castillo, A.: Bitcoin: A primer for policymakers. Mercatus Center at George Mason University (2013)
5. Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2013)
6. Gastwirth, J.L., Bhattacharya, P.K.: Two probability models of pyramid or chain letter schemes demonstrating that their promotional claims are unreliable. Operations Research 32(3), 527–536 (1984)
7. Juels, A., Kosba, A.E., Shi, E.: The Ring of Gyges: Investigating the future of criminal smart contracts. In: ACM CCS. pp. 283–295 (2016)
8. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: ACM CCS (2016), http://eprint.iacr.org/2016/633
9. Moore, T.: The promise and perils of digital currencies. IJCIP 6(3-4), 147–149 (2013)
10. Moore, T., Han, J., Clayton, R.: The postmodern Ponzi scheme: Empirical analysis of high-yield investment programs. In: Financial Cryptography and Data Security. pp. 41–56 (2012)
11. Murphy, E.V., Murphy, M.M., Seitzinger, M.V.: Bitcoin: Questions, answers, and analysis of legal issues. Tech. rep., Congressional Research Service (2015)
12. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
13. Seijas, P.L., Thompson, S., McAdams, D.: Scripting smart contracts for distributed ledger technology. Cryptology ePrint Archive, Report 2016/1156 (2016), http://eprint.iacr.org/2016/1156
14. Slattery, T.: Taking a bit out of crime: Bitcoin and cross-border tax evasion. Brook. J. Int’l L. 39, 829 (2014)
15. Szabo, N.: Formalizing and securing relationships on public networks. First Monday 2(9) (1997), http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/548
16. Vasek, M., Moore, T.: There’s no free lunch, even using Bitcoin: Tracking the popularity and profits of virtual currency scams. In: Financial Cryptography and Data Security. pp. 44–61 (2015)
17. Vasek, M., Thornton, M., Moore, T.: Empirical analysis of denial-of-service attacks in the Bitcoin ecosystem. In: Financial Cryptography and Data Security. pp. 57–71 (2014)
18. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. gavwood.com/paper.pdf (2014)

19. Yujian, L., Bo, L.: A normalized Levenshtein distance metric. IEEE Trans. on Pattern Analysis and Machine Intelligence 29(6), 1091–1095 (2007)