Understanding Ethereum via Graph Analysis

TING CHEN and ZIHAO LI, Center for Cybersecurity, University of Electronic Science and Technology of China, China
YUXIAO ZHU, School of Management, Guangdong University of Technology; School of Big Data and Strategy, Guangdong University of Technology, China
JIACHI CHEN and XIAPU LUO, Department of Computing, The Hong Kong Polytechnic University, China
JOHN CHI-SHING LUI, Department of Computer Science & Engineering, The Chinese University of Hong Kong, China
XIAODONG LIN, University of Guelph, Canada
XIAOSONG ZHANG, Center for Cybersecurity, University of Electronic Science and Technology of China, China

18

Ethereum, a blockchain, supports its own cryptocurrency named Ether and smart contracts. Although more than 8M smart contracts have been deployed on Ethereum, little is known about the characteristics of its users, smart contracts, and the relationships among them. We conduct the *first* systematic study on Ethereum by leveraging graph analysis to characterize three major activities on Ethereum, namely money transfer, smart contract creation, and smart contract invocation. We collect all transaction data, construct three graphs from the data to characterize major activities via graph analysis, and discover new insights. Moreover, we address three security issues based on graphs.

CCS Concepts: • **Networks** → Network monitoring; • **Security and privacy** → *Distributed systems security*;

Additional Key Words and Phrases: Blockchain, Ethereum, money flow graph, contract creation graph, contract invocation graph, graph analysis

ACM Reference format:
Ting Chen, Zihao Li, Yuxiao Zhu, Jiachi Chen, Xiapu Luo, John Chi-Shing Lui, Xiaodong Lin, and Xiaosong Zhang. 2020. Understanding Ethereum via Graph Analysis. *ACM Trans. Internet Technol.* 20, 2, Article 18 (April 2020), 32 pages.
https://doi.org/10.1145/3381036

Ting Chen is partially supported by the National Natural Science Foundation of China (61872057) and National Key R&D Program of China (2018YFB0804100). Xiapu Luo is partially supported by Hong Kong RGC Project (No. 152193/19E).
Authors' addresses: T. Chen, Z. Li, and X. Zhang, Center for Cybersecurity, University of Electronic Science and Technology of China, No.2006, Xiyuan Road, West High Technology District, Chengdu, Sichuan, China, 611731; emails: brokendragon@uestc.edu.cn, gforiq@qq.com, johnsonzxs@uestc.edu.cn; Y. Zhu, School of Management, Guangdong University of Technology; School of Big Data and Strategy, Guangdong University of Technology, Guangzhou, 100 Waihuan Xi Road, Panyu District, Guangzhou, Guangdong Province, China, 510006; email: zhuyuxiao.mail@gmail.com; J. Chen and X. Luo (corresponding author), Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China; emails: {csjcchen, csxluo}@comp.polyu.edu.hk; J. C.-S. Lui, Department of Computer Science & Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong; email: cslui@cse.cuhk.edu.hk; X. Lin, University of Guelph, 50 Stone Road East, Guelph, Ontario, Canada N1G 2W1; email: xlin08@uoguelph.ca.

1533-5399/2020/04-ART18 $15.00
https://doi.org/10.1145/3381036

1 INTRODUCTION

With more than 10B USD market capitalization [2], Ethereum is the largest blockchain that runs smart contracts. A smart contract is an autonomous computer program that, once started, executes automatically and mandatorily according to the program logic defined beforehand [28]. The smart contracts on Ethereum are usually developed by any high-level language (e.g., Solidity) and then compiled into bytecode. After being deployed on Ethereum, the bytecode will run on the blockchain when it is invoked. Since its debut on July 30, 2015, there are already around 40M external owned accounts, which are usually normal users or developers, and 8M smart contracts in Ethereum. There are various studies about Ethereum's security [16, 24, 32, 33, 43] and performance [15, 17]. However, little is known about the characteristics of its users, smart contracts, and the relationships among them (i.e., user to user, user to smart contract, smart contract to smart contract), which can empower us to better understand the Ethereum ecosystem.

In this article, we conduct the *first* systematic study on Ethereum by leveraging graph analysis to characterize three major activities on Ethereum: money transfer, smart contract creation, and smart contract invocation. These activities empower users to send money to others, developers to deploy their smart contracts on Ethereum, and users or applications to call the deployed smart contracts. We construct the money flow graph (MFG), smart contract creation graph (CCG), and smart contract invocation graph (CIG) to characterize these activities by collecting *all* transactions that happened on Ethereum from July 30, 2015, to November 1, 2018, and extracting useful data from them.

Transactions are signed data packages containing messages with useful information. For example, the *value* field in a transaction indicates the amount of money transferred. A transaction can be an external one if it is sent from an external owned account (EOA), which is usually produced by a *wallet* application with data provided by the user (e.g., how much money to transfer). Alternatively, a transaction can be an internal one that results from executing a smart contract due to an external transaction, and therefore an internal transaction's sender is the smart contract. Note that an external transaction may lead to many internal transactions. It is non-trivial to collect all transactions, because although external transactions are publicly available in the blockchain, internal transactions are not stored in the blockchain. To obtain all internal transactions, we design a new approach that replays all external transactions in our instrumented Ethereum client that will record the internal transactions (Section 4).

Based on all transaction data, we construct MFG, CCG, and CIG to represent money flow, smart contract creation, and smart contract invocation, respectively (Section 5). Then, we conduct various graph analysis on MFG, CCG, and CIG, such as measuring their degree distribution, clusters, degree correlation, node importance, assortativity, strongly/weakly connected component (SCC/WCC), and the evolution of these graphs by investigating the changing of metrics over time (Section 6). Such investigation leads to new observations and insights, and we present some of them in this section. First, users prefer transferring money rather than calling smart contracts on Ethereum. A possible reason is that many users of Ethereum have experiences on Bitcoin or other blockchains to transfer cryptocurrencies, but they may be unfamiliar with smart contracts. Second, smart contracts for financial applications dominate the Ethereum ecosystem, but most smart contracts are not frequently invoked. A reason may be that financial applications are naturally favored by blockchains because of cryptocurrencies, but extending blockchains to other application domains is still in active exploration.

Third, the trend that the accounts in MFG tend to transfer Ether rather than deposit Ether holds at all times. Such a trend is accordant with the active speculative behaviors that receive and send ETH frequently to earn money. Fourth, the trend that if an account creates lots of contracts, then

the created contracts do not tend to create new contracts holds at all times. We find that the huge number of created contracts by the same account is similar and do simple tasks that do not need to create other smart contracts. Fifth, we find huge SCCs in MFG, indicating hub nodes that may be exchange markets. Sixth, most WCCs of CCG are small, suggesting that most applications just consist of a few smart contracts. Seventh, the clustering coefficient of CIG approaches 0. One possible reason is that most applications on Ethereum are not complicated, and hence they do not require intensive interaction between smart contracts.

Besides examining individual graphs, we propose new approaches based on the three graphs to address three important security issues in Ethereum (Section 7), including attack forensics for finding accounts controlled by the attacker, anomaly detection for discovering abnormal accounts that create lots of unused contracts, and deanonymization for revealing the identities of accounts. The experimental result demonstrates the effectiveness of our new approaches.

There are a few studies investigating Ethereum by graph analysis [12–14, 30, 41]. The existing studies differ with our work in many aspects. More importantly, they neither collect the transactions from EOAs to contracts, nor measure the data by graph techniques. For example, Chan and Olmsted focus on fast query by storing the graph into a graph database [13]. Somin et al. focus on a special contract behavior, so-called token transfer, which is orthogonal with our work [41]. Kiffer et al. model the creation and invocation of smart contracts as a graph [30]. Although many recent works studied Bitcoin through graph analysis[7, 23, 34, 36, 39, 47, 48], their methods and results cannot be directly applied to Ethereum because of the differences between Ethereum and Bitcoin in functionalities and protocols, as discussed in Section 8.1.

This article extends our previous conference paper [18] in *six* aspects. First, the journal version presents the observations and insights based on more comprehensive data. In particular, the journal version collects all transactions till November 1, 2018, which is about 14× more than the data analyzed in the conference version. Second, the new version of Ethereum introduces a new operation STATICCALL that can also result in internal transactions. We handle this new operation in this manuscript to collect complete internal transactions. Third, we implement a new application, deanonymization based on graph analysis. Fourth, we apply our anomaly detector to *all* smart contracts (>8M) and discover more (i.e., 48) abnormal accounts, while the conference version just analyzes the top 10 most important accounts of CIG. By analyzing these abnormal accounts, we find new attacks that stole money. Fifth, we analyze the strongly connected components (SCCs) and weakly connected components (WCCs) of CIG in the journal version, whereas the conference version did not do it. Last, we present the evolution of the three graphs over time and have interesting observations.

In summary, we make the following major contributions:

(1) To the best of our knowledge, it is the *first* systematic investigation on Ethereum via graph analysis. We propose a new approach to collect all transaction data and then construct money flow graph (MFG), smart contract creation graph (CCG), and smart contract invocation graph (CIG) to characterize major activities on Ethereum.
(2) We obtain many new observations and insights about Ethereum by evaluating various graph metrics (e.g., degree distribution) on MFG, CCG, and CIG. They empower us to obtain a better understanding of the Ethereum ecosystem. We list 35 major observations from our analysis in Table 1.
(3) We propose new approaches based on the three graphs to handle three important security issues in Ethereum, including attack forensics, anomaly detection, and deanonymization. The evaluation through real cases shows the effectiveness of our new approaches.

Table 1. 35 Major Observations from Our Analysis

Section	Observation
5.1	Only 0.2% EOAs do not transfer any Ether.
	More than 83% smart contracts (the point (0, 0.83)) do not transfer any Ether.
	About 85% accounts (96% smart contracts and 82.5% EOAs) are involved in no more than 5 transactions.
	99.7% EOAs do not create contracts.
	99.2% of smart contracts are involved in only one transaction, i.e., the transaction for contract creation).
	66% EOAs do not invoke smart contracts.
	47% smart contracts are not invoked.
	About 92% EOAs call smart contracts no more than 5 times.
5.2	88% edges in MFG are between EOAs.
5.4	84% edges in CIG are from EOAs to smart contracts.
6.2	The degree/indegree/outdegree distributions of MFG follow power law.
	Ether distribution conforms to the power law, which is fairer than degree distribution.
	The degree of nearly 93% accounts is no larger than 6.
	If two accounts A, B trade with a third account C, A and B are likely to trade with each other.
	Whether there exists an edge between two nodes is not determined by the degrees of the two nodes.
	A node with a large indegree is likely to have large outdegree and vice versa.
	Hub nodes, which may be exchange markets, exist in MFG.
	Exchange markets are the most important nodes in MFG.
	Users do not usually transfer much money by a single transaction.
	Users prefer to send 100 and 1,000 Ether.
	Almost all Ether transfers are not from contract creators to their created contracts.
	Almost all smart contracts do not receive ETH from their creators.
	Almost all Ether transfers are not to the invoked smart contracts.
	Contract callers do not often transfer Ether from/to the called smart contracts.
6.3	The degree distribution of CCG follows the power law.
	Large-degree nodes of CCG tend to connect small-degree nodes.
	One developer directly or indirectly creates 17.2% of all smart contracts on Ethereum.
	Most applications just consist of a few of smart contracts.
	74% of smart contracts are not invoked by their creators.
	Most contract invocations are not from contract creators to the smart contracts created by them.
6.4	The degree/indegree/outdegree distributions of CIG follow the power law.
	Whether a smart contract invokes another smart contract is not determined by its degree.
	If an account A calls contract B and C, then B and C are very unlikely to call each other.
	How nodes connect with each other in CIG is not determined by the degree of nodes.
	Token contracts are important nodes in CIG.

The rest of the article is organized as follows: Section 2 introduces the background knowledge of Ethereum. After giving an overview of our analysis procedure in Section 3, we detail the data collection, graph construction, and analysis in Section 4, Section 5, and Section 6, respectively. Section 7 presents the new approaches based on cross-graph analysis for handling two security issues. After reviewing related studies in Section 8, we conclude the article in Section 9.

2 BACKGROUND

Accounts. The basic unit in Ethereum is the *account* [3]. There are two kinds of accounts: external owned accounts (EOAs) and smart contracts. The major difference between them is that only smart contracts contain executable code that is developed by users [3]. Developers can first develop smart

contracts in high-level programming language to provide various services and compile them into bytecode before deployment. A smart contract can be created by an EOA or a smart contract, but an EOA cannot be created by another account [46].

Transactions. A transaction is a message sent from one account to another. The interaction of accounts, such as the activities studied by our work, is accomplished by transactions. There are two types of transactions: the *external transaction* sent by an EOA and the *internal transaction* sent by a smart contract. Both the external transaction and internal transaction can be used for money transfer, contract creation, and contract invocation, and hence, we handle both. If a transaction attempts to create a smart contract, its *receiver* field should be *null* [46]. If the smart contract is successfully created, then the address of the created smart contract will be returned [46]. Contrarily, if a transaction attempts to invoke a smart contract, its receiver field should be the address of the callee [46]. An external transaction has a similar data structure with an internal transaction, including the receiver field, the amount of transferred ETH, and the input data that carry the EVM bytecode of a smart contract to be deployed or the parameters for invoking a smart contract [46]. The major difference between them is that only the former is sent by an EOA and recorded in the blockchain, while the latter is sent by a smart contract and not recorded in the blockchain.

Ether/Token. Being the native cryptocurrency of Ethereum, Ether can be traded on cryptocurrency exchanges or transferred among accounts. Users transfer Ether to others by sending transactions whose *value* field indicates the amount of Ether transferred. Besides, Ethereum requires users to pay transaction fees in Ether for protecting the blockchain from frivolous or malicious tasks that will exhaust the resources [46]. The account who produces blocks (i.e., miners) will be rewarded in Ether. Since mining is difficult, many miners form a mining pool to increase the probability of producing blocks. If a block is produced by a mining pool, then the miners of the pool will share the reward. Token is a kind of cryptocurrency implemented as smart contracts [3]. Token can also be traded and transferred.

Smart contract. Developers can create various applications on Ethereum by constructing and deploying smart contracts. More specifically, developers usually prepare smart contracts in high-level languages (e.g., Solidity) and compile them into bytecode. To deploy a smart contract on Ethereum, the developer sends a transaction, whose *data* field contains the bytecode rather than the source, to the *null* address, indicating contract creation. Note that a smart contract can be created by either EOA or another smart contract. After a successful deployment, the address of the contract will be generated. Any user of Ethereum can invoke the smart contract by sending a transaction whose *recipient* field is the contract address. The *data* field of that transaction specifies the function in the smart contract to be invoked as well as function's parameters. Smart contracts are executed in Ethereum virtual machine (EVM), which is a stack-based virtual machine.

Ethereum client. The underlying topology of Ethereum blockchain is a P2P network. Each peer runs an Ethereum client, which is responsible for synchronization with its peers. The client contains an EVM for executing smart contracts. By synchronization, a client downloads all blocks from other peers and constructs the blockchain in the local machine by replaying all historical transactions in the EVM of this client.

3 METHODOLOGY

As shown in Figure 1, our methodology consists of three phases, which are described in the following sections. The first phase, *data collection* (Section 4), collects *all* transaction data for subsequent graph construction. Note that although the blockchain and all external transactions are publicly available, the internal transactions are not stored in the blockchain. We propose a novel approach

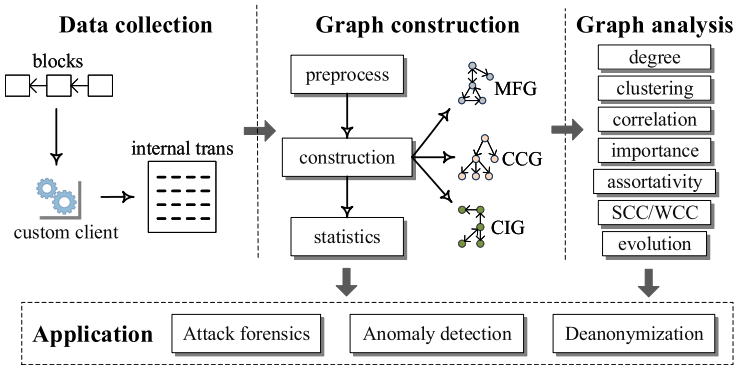


Fig. 1. An overview of our approach. Our approach consists of three phases, and we implement three applications based on graph analysis.

to obtain internal transactions by instrumenting the Ethereum client to add recording code. Then, both external and internal transactions are fed to the second phase, *graph construction* (Section 5), to construct three graphs: money flow graph (MFG), smart contract creation graph (CCG), and smart contract invocation graph (CIG). These graphs characterize the activities of money transfer, contract creation, and contract invocation, respectively. We also calculate the statistics of these graphs.

The third phase, *graph analysis* (Section 6), conducts graph analysis on MFG, CCG, and CIG by computing metrics including degree distribution, clustering, degree correlation, node importance, assortativity, and strongly/weakly connected component. We also analyze the evolution of these graphs over time in this phase. Based on the statistics and metrics of these graphs, we obtain new observations and insights. Besides inspecting individual graphs, we propose new approaches based on those graphs to handle security issues in Ethereum (Section 7), including attack forensics, anomaly detection, and deanonymization.

4 DATA COLLECTION

We collect all accounts (39,572,734 EOAs and 8,725,788 smart contracts) and transactions (336,410,379 external transactions and 333,011,145 internal transactions) from the launch of Ethereum on July 30, 2015, to November 1, 2018, and exclude three types of transactions. The first is the failed transaction. The submission of transactions to the blockchain may fail due to many reasons, e.g., the deployed contract is oversize. Once a transaction fails, the effect resulting from submitting the transaction will be nullified by rollback operations [46]. A transaction of the second type sends Ether from an EOA to another but the amount is zero. A transaction of the third type self-destructs a smart contract and the smart contract has no Ether remaining. Consequently, the last two types of transactions do not result in money transfer.

We do not adopt two alternatives to collect transactions. The first is crawling from some Ethereum explorers (e.g., Etherscan [4]) or invoking the web APIs provided by the Ethereum explorers. However, to keep the websites available to all visitors, they usually restrict the amount of data that can be crawled or downloaded. For example, the maximum number of external transactions that the web API of Etherscan allows users to download is 10K [21]. As another example, Etherscan just displays the last 0.5M external transactions that account for about 0.15% [4]. The second is invoking the web3 API, `web3.eth.getTransaction()` provided by an Ethereum client [20]. Unfortunately, the API cannot obtain internal transactions. Moreover, users need to

provide the transaction hash when invoking the API. However, there is no easy way to obtain the transaction hashes of all transactions. We describe our instrumentation-based approach to collect both external transactions and internal transactions as follows.

4.1 External Transactions Collection

Since external transactions are sent by EOAs and stored in the blockchain, we collect them by running an Ethereum client to synchronize all data. Note that each Ethereum client maintains the same copy of blockchain with all historical transactions according to its protocol [46]. Technically, we instrument the official Ethereum client, Geth. The function `TransitionDb()` in \core\state_transition.go is responsible for submitting external transactions. It calls `vmenv.Create()` if the transaction attempts to create a new contract; otherwise, it calls `vmenv.Call()`. The account creating the contract and the amount of Ether sent to the newly created contract are the parameters of `vmenv.Create()`. Whether the contract is successfully created and the address of the created contract can be found in the return values of `vmenv.Create()`. Besides the addresses of caller and callee and the amount of Ether sent to the callee are the parameters of `vmenv.Call()`. Whether contract invocation is successful is specified in the return values of `vmenv.Call()`. Therefore, we collect external transactions by inserting recording code to `TransitionDb()`.

4.2 Internal Transactions Collection

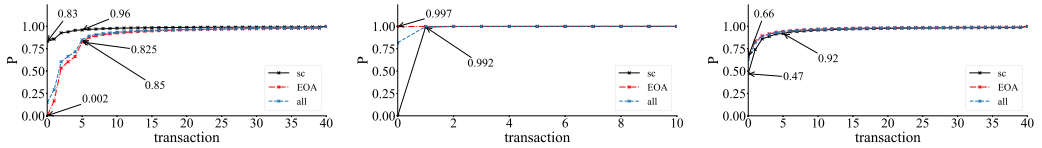
Internal transactions result from the execution of smart contracts, but they are not stored in the blockchain. One should not ignore internal transactions when conducting graph analysis, because they enable smart contracts to interact with other accounts, such as creating new contracts, invoking other contracts. To address this issue, we propose to collect all internal transactions when EVM replays all historical external transactions. The rationale behind our solution is that internal transactions are incurred by the execution of smart contracts, which are initialized by external transactions. Each node of Ethereum will replay all external transactions inside its EVM for achieving consensus with other nodes [46], and hence each node will reproduce all historical internal transactions.

To record internal transactions, we add the code for recording into the EVM, which is open-source. More specifically, EVM defines 100+ operations for EVM, but only 6, which are CREATE, CALL, CALLCODE, DELEGATECALL, STATICCALL, and SELFDESTRUCT (an alias of the original SUICIDE operation) can generate internal transactions [46]. CREATE and CALL create and invoke a smart contract, respectively [46]. CALLCODE and DELEGATECALL also invoke a smart contract, but the callee runs in caller's context [46]. By using them, a smart contract can be loaded as a library by another one. When calling a contract by STATICCALL, the state (e.g., the amount of Ether, a global variable) of the callee cannot be modified [46]. SELFDESTRUCT removes the smart contract from the blockchain and sends the remaining money in the smart contract to a designated target. Therefore, we insert recording code into the handlers of these six operations.

EVM prepares handlers for interpreting all EVM operations, so we just need to add recording code into the handler of those 6 EVM operations. In the handler of CREATE, we add the recording code after the accomplishment of contract creation to record the address of contract creator, the address of the created contract, and the amount of Ether deposited in the created contract by the contract creator. Similarly, in the handlers of CALL, CALLCODE, DELEGATECALL, and STATICCALL, we log the addresses of caller, callee, and the amount of Ether transferred from the caller to the callee. In the handler of SELFDESTRUCT, we record the address of the self-destructed contract, the address to receive the remaining Ether, and the amount of Ether.

Table 2. Number of (External and Internal) Transactions in Each Group

	BG_{MF}	BG_{CC}	BG_{CI}	$BG_{MF} \cap BG_{CC}$	$BG_{MF} \cap BG_{CI}$	$BG_{CC} \cap BG_{CI}$
External	214,922,287	6,039,698	169,041,592	23,345	29,347,610	0
Internal	22,242,240	2,686,090	221,379,944	462,814	7,762,082	0
Total	237,164,527	8,725,788	390,421,536	486,159	37,109,692	0



(a) BG_{MF} (b) BG_{CC} (c) BG_{CI}

Fig. 2. Cumulative distribution of the number of transactions.

4.3 Account Collection

After collecting transactions, the collection of accounts is straightforward, because each transaction associates with an account for sending and an account for receiving. Then, we recognize a smart contract by checking the receiver field of a transaction. Particularly, if the receiver field is *null*, then the return address indicates a smart contract. After recognizing smart contracts, the remaining accounts are EOAs.

5 GRAPH CONSTRUCTION

Money transfer, contract creation, and contract invocation are three major activities happening on Ethereum. To investigate them, we construct three graphs (i.e., MFG, CCG, CIG) based on both external and internal transactions.

5.1 Transaction Preprocessing

From all collected transactions, we obtain 48,298,522 accounts, including 39,572,734 EOAs and 8,725,788 smart contracts in total. Note that the accounts are recorded by client instrumentation (Section 4.3). In the *preprocessing* stage, we divide all collected transactions into three groups (i.e., BG_{MF}, BG_{CC}, and BG_{CI}), corresponding to money transfer, smart contract creation, and smart contract invocation for subsequent graph construction. Please note that a transaction may be included in several groups if it triggers multiple activities. For instance, a transaction for smart contract creation can also deposit some Ether in the newly created smart contract and therefore the transaction should be included in both BG_{MF} and BG_{CC}. Similarly, when a transaction calls a smart contract, it can also send Ether to the contract, and thus such transaction should be included in BG_{MF} and BG_{CI}. It is impossible that a transaction belongs to both BG_{CC} and BG_{CI}, because a smart contract should be created by a transaction and then invoked by a subsequent transaction. Table 2 lists the number of transactions including external transactions and internal transactions involved in money flow (BG_{MF}), contract creation (BG_{CC}), contract invocation (BG_{CI}), both BG_{MF} and BG_{CC}, both BG_{MF} and BG_{CI}, and both BG_{CC} and BG_{CI}, respectively. It shows that the size of BG_{CC} (8,725,788) is equal to the number of smart contracts, because a smart contract can only be created once.

Figure 2 illustrates the distribution of the number of transactions related to the accounts in BG_{MF}, BG_{CC}, and BG_{CI}, individually. The dashed line, dash-dot line, and solid line with marks

denote all accounts, EOAs, and smart contracts, respectively. Each point (x, y) indicates that y of accounts are involved in no more than x transactions. Figure 2(a) shows that 0.2% (the point (0, 0.002)) EOAs do not transfer any Ether. In other words, if an EOA is a user, then almost all users transfer (i.e., receive or send) money on Ethereum. Such observation is expected, because the easiest way to use Ethereum is sending or receiving ETH. However, more than 83% smart contracts (the point (0, 0.83)) do not transfer any Ether. A possible reason is that the program logic of those smart contracts may not involve ETH transfer. For example, the standard methods defined in the ERC-20 standard (the most popular token standard) support token transfer, querying the amount of tokens, and so on, but they are not designed for ETH transfer [44]. Moreover, about 85% accounts (96% smart contracts and 82.5% EOAs) are involved in no more than five transactions; that is, most accounts, especially smart contracts, did not frequently transfer ETH. Such observation may suggest that Ethereum, a new blockchain-based computation platform, has not become a necessity of daily life yet.

Figure 2(b) shows that 99.7% EOAs (point (0, 0.997)) do not create contracts. If an EOA that creates smart contracts is a smart contract developer, then the proportion of developers is just 0.3% total users. The result is unsurprising, because application users always outnumber application developers. The point (1, 0.992) indicates that 99.2% of smart contracts are involved in only one transaction, i.e., the transaction for contract creation. Hence, we learn that almost all contracts do not create contracts. One possible reason is that Ethereum is such a young platform that developers rarely exploit this advanced functionality. Figure 2(c) demonstrates that 66% (point (0, 0.66)) EOAs do not invoke smart contracts and 47% (point (0, 0.47)) smart contracts are not invoked. Moreover, about 92% EOAs call smart contracts no more than five times (point (5, 0.92)). Hence, we can learn that most users use smart contracts infrequently. A reasonable explanation is that most smart contracts do not belong to hot applications; instead, they may be toy examples deployed by Ethereum fans. Since 83% of smart contracts do not transfer money, we investigate whether they are invoked. Results show that 57% $(4,107,460/7,233,706)$ of these smart contracts are not invoked (i.e., those smart contracts are not used at all) and thus considerable resources (e.g., network, disk) are wasted to synchronize and store them.

5.2 MFG Construction

Definition 5.1. $MFG = (V, E, w)$, where V is a set of nodes, E is a set of edges and w is a function mapping edges to their weights. $V = V_n \bigcup V_{sc}$, V_n is the set of EOAs, and V_{sc} is the set of smart contracts. E is a set of *ordered* pairs of nodes, $E = \{(v_i, v_j) | v_i, v_j \in V\}$. The order of an edge indicates the direction of transferred money. $w : E \rightarrow \mathbb{R}_+$ associates each edge with a weight, which is the total amount of transferred Ether along the edge by one or more transactions. In summary, MFG is a weighted directed graph.

We use the terms *account* and *node* interchangeably in the remainder of this article. To construct MFG, we iteratively process the transactions in BG_{MF}. In particular, for every transaction, the accounts who send or receive Ether are nodes, and we connect an edge whose weight is the amount of transferred Ether from the sender to the recipient. If the edge has already existed, then we add its weight by the amount of transferred Ether. Note that the sender, recipient, and the amount of Ether are recorded by client instrumentation (Section 4). Table 3 lists the statistics of MFG, including the number of edges, the number of isolated EOA, and the number of isolated smart contracts (*sc*). An isolated node in MFG means that it neither receives nor sends Ether. Only 88,361 EOAs (compared to 39,572,734 in total) do not transfer money, since they are isolated in MFG. The data in Table 3 match the observations from Figure 2(a) that almost all EOAs transfer money, but more than 80% of smart contracts do not. We classify the edges of MFG into four kinds according to senders and

Table 3. Statistics of Graphs

graph	# edges	# isolated EOA	#isolated sc
MFG	95,963,113	88,361	7,233,706
CCG	8,725,788	39,464,113	0
CIG	38,708,166	26,301,754	4,121,492

Table 4. Edges of MFG

sender \ receiver	EOA	sc
EOA	84,704,702	6,653,306
sc	3,603,039	1,002,066

Table 5. Edges of CCG

sender \ receiver	EOA	sc
EOA	0	6,039,698
sc	0	2,686,090

Table 6. Edges of CIG

sender \ receiver	EOA	sc
EOA	0	32,435,514
sc	0	6,272,652

receivers. Results are reported in Table 4. We find that 88% (84, 704, 702/95, 963, 113) of edges are between EOAs, which is accordant with Figure 2(a) that almost all EOAs transfer ETH but most smart contracts did not do it.

5.3 CCG Construction

Definition 5.2. $CCG = (V, E)$, where V is a set of nodes, the same as those in Definition 5.1 (in revised paper). E is a set of edges. $E = \{(v_i, v_j)|v_i \in V, v_j \in V_{sc}\}$, where a directed edge (v_i, v_j) from v_i to v_j indicates that an account v_i creates a smart contract v_j.

The definition implies several properties of CCG. First, if $(v_i, v_j) \in E$, then $(v_j, v_i) \notin E$. That is, the edge between two nodes is unidirectional, because if a node A creates a contract B, then B cannot create A. Second, if $(v_i, v_j) \in E$, then $(v_k, v_j) \notin E, \forall k \neq i$. Since a smart contract cannot be created twice and an EOA cannot be created by another account, CCG contains no cycles. Therefore, we can view CCG as a forest consisting of multiple trees. The root of each tree is an EOA, and the other nodes of the tree are smart contracts directly or indirectly created by the root. Since each transaction in BG_{CC} indicates the creation of a smart contract, we add an edge from the transaction sender to the created smart contract.

The statistics of CCG (Table 3) match the observations from Figure 2(b) that the vast majority of EOAs are isolated, i.e., they do not create contracts. Moreover, the smart contracts (8,725,788) obviously outnumber the EOAs (i.e., $108,621 = 39,572,734 - 39,464,113$), which create contracts. If an EOA that creates smart contracts is a developer, then the number of developers is much fewer than that of smart contracts. In practice, the difference may be more significant, since a developer can have many EOAs. We classify the edges in CCG into two kinds: from EOAs to smart contracts and from smart contracts to smart contracts, because only smart contracts can be created, as shown in Table 5. The results in Table 5 are accordant with the results in Table 2, because each edge in CCG corresponds to a unique transaction (i.e., the weight of each edge is 1).

5.4 CIG Construction

Definition 5.3. $CIG = (V, E, w)$, where V is a set of nodes, E is a set of edges, and w is a function mapping edges to their weights. E is an *ordered* pairs of nodes, $E = \{(v_i, v_j)|v_i \in V, v_j \in V_{sc}\}$. An edge (v_i, v_j) indicates that the caller v_i invokes the callee v_j. $w : E \to \mathbb{N}$ associates each edge with a weight, which is the total number of invocations along the edge by one or more transactions. Hence, CIG is a weighted directed graph.

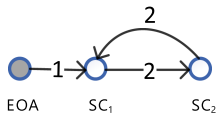


Fig. 3. An example to explain high weights of edges between smart contracts.

To construct CIG, we process every (external and internal) transaction in BG_{CI} to get the callers' and callees' addresses, which are the nodes in CIG. Then, we add an edge from sender to recipient and set the edge's weight to 1 if there is no edge between them. Otherwise, we increase the edge's weight by 1. Table 3 shows the statistics of CIG, which match the observations from Figure 2 that 66% (26, 301, 754/39, 572, 734) of EOAs do not call smart contracts, and 47% (4, 121, 492/8, 725, 788) of smart contracts are not invoked. We classify the edges in CIG into two kinds: from EOAs to smart contracts and from smart contracts to smart contracts, because only smart contracts can be invoked, as shown in Table 6. We find that 84% (32, 435, 514/38, 708, 166) of edges are from EOAs to smart contracts. Such a high proportion is consistent with Table 2 that there are more internal transactions than external transactions in CIG, because we find that the average weight (i.e., the number of transactions) of the edges from smart contracts to smart contracts is higher than the average weight of the edges from EOAs to smart contracts. The high weights of the edges between smart contracts may result from the program logic of smart contracts. For example, if an EOA calls a smart contract, SC_1, which invokes SC_2, and SC_2 calls back to SC_1, which invokes SC_2 again, and finally SC_2 calls back to SC_1 again, as shown in Figure 3, the weight of the edge from the EOA to SC_1 is one and the weights of the two edges between SC_1 and SC_2 are two.

We obtain the following insights by building these graphs:

Insight 1. Users prefer to transfer money on Ethereum instead of using smart contracts. One possible reason is that many users of Ethereum may have experiences in using Bitcoin or other cryptocurrency blockchains. However, smart contracts may be relatively new to them.

Insight 2. The smart contracts are not widely used. One possible reason is that (as shown in Section 6.4) there is a limited number of applications supported by smart contracts and most of them are for financial applications. Financial applications are naturally favored by blockchains because of cryptocurrencies, but extending blockchains to other applications is still in active exploration.

Insight 3. Not all users frequently use Ethereum. One possible reason is that most users just try Ethereum and deploy toy contracts on it.

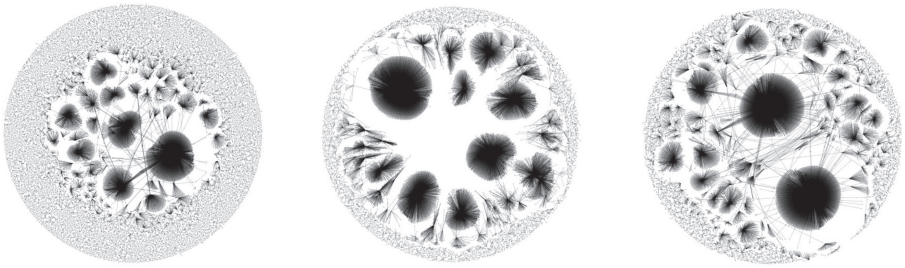
Figure 4 visualizes the three graphs. They are different and have noticeable structure features. We can find several community structures in them, indicating the existence of a few large-degree nodes and many small-degree nodes. In other words, a few accounts play a vital role in Ethereum. We investigate the structures of three graphs in Section 6.

6 GRAPH ANALYSIS

This section investigates MFG, CCG, and CIG from various metrics in graph analysis. Please note that we do not consider isolated nodes when computing the metrics. We first introduce the metrics and then detail the observations from each graph in the following subsections.

6.1 Metrics

Degree distribution. The *degree* of a node is the number of edges connecting to the node, and the *degree distribution* is the fraction P_k of nodes with degree k, for all k [45]. Since P_k is a fraction, $0 \leq$



(a) MFG (b) CCG (c) CIG

Fig. 4. Visualization of MFG, CCG, and CIG. For the ease of illustration, we randomly select 20K edges from each graph to draw the figure.

$P_k \leq 1$ and $\sum P_k = 1$. Specifically, the degree of a node in MFG indicates the number of accounts trading with that node. The degree of an EOA in CCG represents the number of contracts created by it, while the degree of a smart contract in CCG indicates the number of contracts created by it plus 1 (for the transaction to create that contract). The degree of an EOA in CIG is the number of smart contracts invoked by it, while the degree of a contract in CIG is the total number of contracts invoking it and the number of contracts it invokes. Let γ be the slope of the distribution; a higher γ represents a fairer degree distribution. γ is between -2 and -3 for many real-world graphs [19], and that is the case for MFG, CCG, and CIG. The *indegree* of a node in a directed graph is the number of edges whose heads end at that node. Specifically, the indegree of an account in MFG is the number of accounts sending money to it. The indegree of an EOA in CCG and CIG is 0, since it cannot be created or invoked by other accounts according to the definitions of CCG and CIG (Definition 5.2, Definition 5.3). The indegree of a contract in CCG is 1, because it can only be created once. The indegree of a contract in CIG is the number of accounts invoking it. The *outdegree* of a node in a directed graph is the number of edges whose tails end at that node. In particular, the outdegree of an account in MFG is the number of accounts receiving money from it. The outdegree of an account in CCG/CIG indicates the number of contracts created/invoked by it.

Clustering coefficient. We compute the *global clustering coefficient* to evaluate the extent to which nodes in a graph tend to cluster together. The global clustering coefficient is the average of local clustering coefficients over all nodes with degree larger than one [40]. The range of the global clustering coefficient is $[0, 1]$, and interested readers can find its mathematical expression in Soffer and Vazquez's work [40]. We obtain many observations using this metric. For example, Section 6.4 shows that the clustering coefficient of CIG approaches to 0, meaning that the collaboration of contracts in CIG is rare.

Assortativity coefficient. We compute the *assortativity coefficient* to measure the level of correlation between the sets of upstream and downstream nodes constituting the edges in a network graph [35]. For example, as shown in Section 6.4, the assortativity coefficient of CIG approaches 0, indicating that method invocation is determined by program logic rather than the degrees of nodes. Interested readers can find its mathematical expression in Meghanathan's work [35]. The range of the assortativity coefficient is $[-1, 1]$ [35]. The studied graph is strongly assortative, weakly assortative, neutral, weakly disassortative, and strongly disassortative, if the assortativity coefficient falls into $[0.6, 1]$, $[0.2, 0.6)$, $(-0.2, 0.2)$, $(-0.6, -0.2]$, and $[-1, -0.6]$, respectively [35].

Pearson coefficient. Pearson coefficient measures the strength and direction of linear relationships between pairs of continuous variables [29]. Interested readers can find its mathematical expression in a tutorial [29]. Let γ be the Pearson coefficient, then $-1 \leq \gamma \leq 1$ [29]. Two pairs of

variables have strong correlation, moderate correlation, weak correlation, very weak correlation, and neural correlation, if $0.5 < |\gamma| \leq 1$, $0.3 < |\gamma| \leq 0.5$, $0.1 < |\gamma| \leq 0.3$, $0 < |\gamma| \leq 0.1$, and $\gamma == 0$, respectively [29]. We use it to evaluate the correlation between the indegree and the outdegree of nodes, because different values of Pearson coefficient indicate different behavior patterns of accounts. For instance, as shown in Section 6.2, the Pearson coefficient of MFG is 0.45, indicating that indegrees and outdegrees of MFG are moderately correlated; that is to say, accounts tend to transfer ETH rather than deposit ETH.

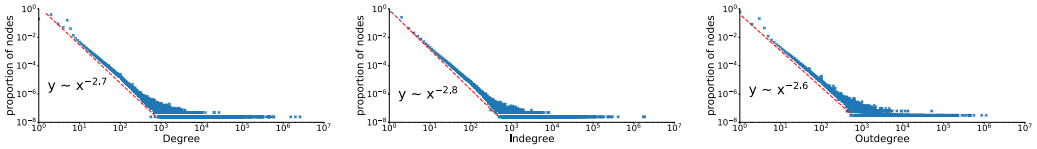
Strongly connected component (SCC). Before explaining the SCC, we define a *path* in a graph as a sequence of edges that connects a sequence of nodes. An SCC of a directed graph G is the set of nodes $C \subseteq V$ such that for every pair of nodes u and v, there is a directed path from u to v and a directed path from v to u [11].

Weakly connected component (WCC). A WCC of a directed graph G is a set of nodes $C \subseteq V$ such that for every pair of nodes u and v, there is an undirected path from u to v [11]; that is, the direction of edges is ignored when looking for WCC.

Importance. We evaluate the *importance* of nodes using two metrics: PageRank and degree centrality. To measure the importance of a node, degree centrality merely considers its degree, while PageRank also considers the importance of its neighbors [10]. Let $PR(x)$ be the PageRank value of node x, then $0 \leq PR(x) \leq 1$ and $\sum PR(x) = 1$, because PageRank forms a probability distribution over nodes [10]. The larger $PR(x)$ is, the more important x is. Interested readers can find the computation procedure of PageRank in Brin and Page's work [10]. By evaluating the importance of nodes, we can discover hot applications. For example, as shown in Section 6.2, the 10 most important nodes in MFG measured by PageRank are exchange markets, highlighting the core position of exchange markets in ETH transfer. We also find that some nodes are regarded as important in terms of both degree centrality and PageRank, although these two metrics do not produce the same results. Such a finding highlights the value of applying more metrics to study graphs, which will be our future work.

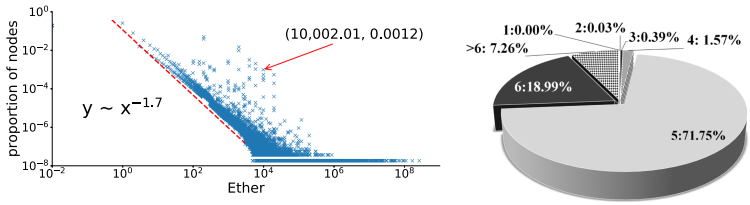
Ether between two accounts (ETA). We propose a new metric, ETA, to describe the average amount of ETH transferred per transaction between two accounts. This metric only applies to MFG, since each edge of MFG indicates ETH transfer between two accounts. To compute ETA, we need the number of transactions for each edge in MFG, which is known when constructing MFG.

Common edges. A common edge between two graphs denotes that two different behaviors happened between two accounts. We use two notions, $E(MFG)$ to represent the edge set of MFG and $|E(MFG)|$ to represent the number of edges of MFG. We define $E(CCG)$, $|E(CCG)|$, $E(CIG)$, and $|E(CIG)|$ in the same way. We use $E(MFG) \cap E(CCG)$ to represent the common edges between MFG and CCG, and let $|E(MFG) \cap E(CCG)|$ denote the number of common edges between MFG and CCG. We define $E(MFG) \cap E(CIG)$, $|E(MFG) \cap E(CIG)|$, $E(CCG) \cap E(CIG)$, and $|E(CCG) \cap E(CIG)|$ in the same way. We define the metric $|E(MFG) \cap E(CCG)|/|E(MFG)|$ to represent the proportion of the common edges between MFG and CCG to all edges of MFG. We define other five metrics $|E(MFG) \cap E(CCG)|/|E(CCG)|$, $|E(CCG) \cap E(CIG)|/|E(CCG)|$, $|E(CCG) \cap E(CIG)|/|E(CIG)|$, $|E(MFG) \cap E(CIG)|/|E(MFG)|$, and $|E(MFG) \cap E(CIG)|/|E(CIG)|$ in the same way. By computing these metrics, we know the frequencies of two different behaviors between two accounts. For example, as shown in Section 6.3, $|E(CCG) \cap E(CIG)|/|E(CCG)| = 0.26$, indicating that most smart contracts have not been invoked by their creators. A possible reason is that the purpose of a contract creator to create smart contracts is providing services to other users rather than herself.



(a) Degree (b) Indegree (c) Outdegree

Fig. 5. Degree/Indegree/Outdegree distributions of MFG. They all follow power law.



(a) Ether distribution (b) An outlier

Fig. 6. Ether distribution and statistics of an outlier.

Evolution. Moreover, we investigate the *evolution* of these graphs over time in terms of the number of nodes, the number of edges, clustering coefficient, Pearson coefficient, assortativity coefficient, the number of SCCs, the size of the largest SCC, the number of WCCs, the size of the largest WCC, and the six metrics related to common edges. To do so, we set a checkpoint at the end of every month after the launching of Ethereum, then we compute the aforementioned metrics based on the data collected before each checkpoint. Therefore, for each metric, we obtain 39 results corresponding to 39 checkpoints. By investigating the evolution, we can discover how Ethereum evolves from its birth.

6.2 MFG Analysis

Degree distribution. Figure 5 shows the degree/indegree/outdegree distributions of MFG, all of which follow the power law, meaning that there are a few large-degree nodes and many small-degree nodes. We also plot the fitting line $y \sim x^{-\alpha}$ for each distribution. The larger the α, the less variable of nodes' degree. As discussed below, these degree distributions match the investigation of important nodes in MFG (Table 9) that a few exchange markets have large degree. The small-degree nodes may be individuals.

We then investigate the distribution of Ether, as shown in Figure 6(a). Each point (x, y) in this figure indicates that there are y of total accounts, and each transfer (i.e., sends and receives) x Ether. Its distribution also conforms to the power law, denoting that a few accounts transfer a lot of money. There are some outliers in Figure 6(a) deviating from the fitted line. Taking them and Figure 5 into consideration, we learn that the outliers are small-degree accounts (i.e., they trade with a few accounts) transferring a lot of money. The slope of Ether distribution (i.e., -1.7) is higher than the slope of degree distribution of MFG (i.e., -2.7), indicating that Ether distribution is fairer than degree distribution. Such observation is expected, because two accounts may transfer different amounts of Ether, even if they transfer Ether to the same number of accounts. Figure 6(b) shows the degree statistics of the accounts corresponding to an outlier, (10,002, 0.0012). The tag $x : y\%$ means that the nodes of degree x account for $y\%$ of total nodes. We find that the degree of nearly 93% accounts is no larger than 6.

Table 7 shows the metrics of MFG. Columns 2–8 list the values of the clustering coefficient, assortativity coefficient, Pearson coefficient, number of SCCs, size (i.e., how many nodes) of the

Table 7. Metrics of the Three Graphs

graph	cluster	assortativity	Pearson	#SCC	largest SCC	#WCC	largest WCC
MFG	0.12	-0.06	0.45	10,012,456	30,878,301	90	40,976,221
CCG	0	-0.29	/	8,834,409	1	81,177	1,501,271
CIG	0.079	-0.01	0.01	17,744,593	103,367	34,284	17,748,461

Table 8. Metrics of Six Metrics Related to Common Edges

$\frac{\lvert E(MFG \cap CCG)\rvert}{\lvert E(MFG)\rvert}$	$\frac{\lvert E(MFG \cap CCG)\rvert}{\lvert E(CCG)\rvert}$	$\frac{\lvert E(CCG \cap CIG)\rvert}{\lvert E(CCG)\rvert}$	$\frac{\lvert E(CCG \cap CIG)\rvert}{\lvert E(CIG)\rvert}$	$\frac{\lvert E(MFG \cap CIG)\rvert}{\lvert E(MFG)\rvert}$	$\frac{\lvert E(MFG \cap CIG)\rvert}{\lvert E(CIG)\rvert}$
0.005	0.06	0.26	0.06	0.08	0.2

largest SCC, number of WCCs, and size of the largest WCC, respectively. Table 8 presents the results of the six metrics related to common edges.

Clustering coefficient. The clustering coefficient is 0.12, revealing that if two accounts A, B trade with a third account C, A and B are likely to trade with each other. A potential reason is that A and B may be friends of C, because there are money flows between A and C, B and C. Therefore, A may also be a friend of B so that there may be money flow between them.

Assortativity coefficient. Its assortativity coefficient approaches to 0, indicating that whether there exists an edge between two nodes is not determined by the degrees of the two nodes. A possible reason is that the users' demand determines money flow. For example, if a user wants to sell ETH, the user will interact with a large-degree account, i.e., an exchange market. On the contrary, if a user wants to send ETH to another individual, the user will interact with a low-degree account, i.e., the receiver.

Pearson coefficient. Pearson coefficient is 0.45, which is moderately large [29], revealing that a node with large indegree is likely to have large outdegree and vice versa. That is, an account will be frequent in both sending and receiving money. Hence, deposit (frequent in receiving but infrequent in sending) is uncommon in Ethereum. Such observation is accordant with the active speculative behaviors that frequently receive and send ETH to earn money.

SCC/WCC. The size of the largest SCC is huge, which contains about 75% nodes ($30,878,301/40,976,455$, 40,976,455 is the number of all accounts minus the number of isolated nodes). It indicates that there should be hub nodes. Such hub nodes may be exchange markets, because they send/receive money to/from a large number of other accounts. The number of SCCs (i.e., 10,012,456) in MFG is far more than that of WCCs (i.e., 90). In this case, a WCC may contain many SCCs, and the money transfer among different SCCs should be unidirectional. Otherwise, the connected SCCs will merge into a bigger SCC. Therefore, if money is transferred from one SCC to another, it never comes back. One possible reason is that some processes or businesses involve several steps. After each step, money is transferred to another account (never used in previous stages) for the next step.

PageRank. Table 9 lists the top 10 most important nodes in MFG, ranked by PageRank, where *sc* indicates smart contract and *PR* denotes PageRank value. Every account is denoted by the first two bytes of its address to save space. Note that the identities of all accounts presented in this article are revealed by our deanonymization application (Section 7.3). If the identities of some accounts cannot be successfully recovered, then we mark their identity and category as "/". Remarkably, all 10 most important nodes in MFG belong to six exchange markets. Moreover, exchange markets, which are hub nodes connecting to other nodes bidirectionally, result in huge SCCs.

Table 9. Top 10 Most Important Nodes of MFG Evaluated by PageRank

account	3f5c	70fa	32be	fbb1	209c	7727	876e	fa52	1151	267b
type	EOA	EOA	EOA	EOA	sc	sc	EOA	sc	EOA	EOA
PR	0.024	0.014	0.0079	0.0078	0.006	0.0058	0.0055	0.0055	0.0049	0.0048
identity	Binance	ShapeShift	Poloniex	Bittrex	Poloniex	Bitfinex	Bitfinex	Kraken	Bitfinex	Kraken
category	exchange	exchange	exchange	exchange	exchange	exchange	exchange	exchange	exchange	exchange

Table 10. Top 10 Most Important Nodes of MFG Evaluated by Degree Centrality

account	3f5c	70fa	fbb1	2b56	32be	8d12	d551	ea67	0681	6090
type	EOA	EOA	EOA	EOA	EOA	sc	EOA	EOA	EOA	sc
degree	2,592,278	1,991,274	1,536,032	590,236	583,150	508,019	456,846	454,884	451,874	442,203
identity	Binance	ShapeShift	Bittrex	KuCoin	Poloniex	EtherDelta	Binance	Ethermine	Binance	name service
category	exchange	exchange	exchange	exchange	exchange	exchange	exchange	mining	Exchange	fundamental

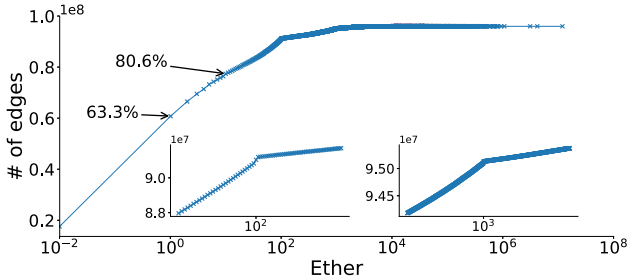
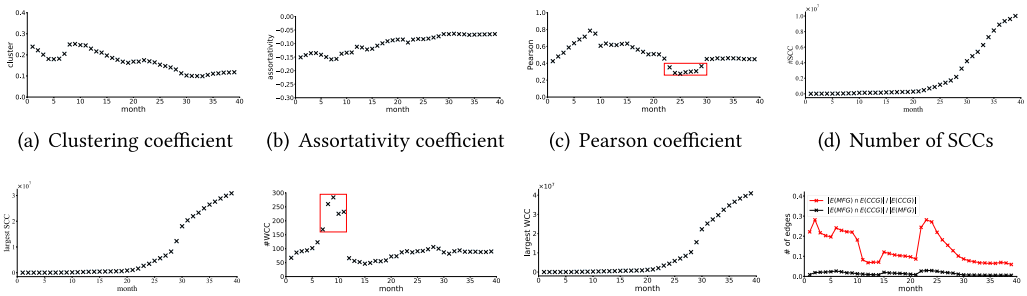


Fig. 7. Ether per transaction between two accounts.

Degree Centrality. Table 10 lists the top 10 most important nodes in MFG, ranked by degree centrality. The columns with grey background indicate that these accounts are also the top 10 most important nodes ranked by PageRank. Eight accounts in this table belong to exchange markets. The account *ea67* belongs to a famous mining pool, Ethermine, which aggregates mining power from a huge number of miners to obtain stable mining rewards. The node *6090* is a smart contract providing name service, through which a user can interact with others by specifying the recipient's name rather than its address. We notice that all smart contracts created by *6090* are open-source and the same as *6090*. In other words, *6090* makes many copies.

ETA. Figure 7 presents the cumulative distribution function (CDF) plot of Ether per transaction between two accounts. x-axis is the amount of Ether, and y-axis is the number of edges in MFG. Please recall that each edge in MFG indicates Ether transfer between two accounts. Each cross (x, y) in this figure denotes that there are y edges in MFG; for each, the average Ether per transaction is no more than x. We find 63.3% and 80.6% of edges in MFG transfer no more than 1 Ether and 10 Ether per transaction, respectively. That means users do not usually transfer much money by a single transaction. Moreover, we observe two turning points where the slope of the curve becomes suddenly smaller. After further analysis, we find that the two turning points correspond to 100 Ether and 1K Ether, respectively. That is, the number of edges suddenly decreases when the average Ether per transaction becomes larger than 100 and 1K. By checking the transactions corresponding to the two turning points, we find that most transactions transfer exactly 100 or 1K Ether. Such observation may be useful in making price decisions.



(a) Clustering coefficient (b) Assortativity coefficient (c) Pearson coefficient (d) Number of SCCs

(e) Size of the largest SCC (f) Number of WCCs (g) Size of the largest WCC (h) $|E(MFG) \cap E(CCG)|$

Fig. 8. Trend of MFG.

Common edges. The proportion of the common edges between MFG and CCG to the total edges of MFG is very low, 0.005, indicating that almost all Ether transfers are not from contract creators to their created contracts. Such observation is accordant with the results presented in Section 5.1 that 99.8% of EOAs transfer ETH, but only 0.3% of EOAs create smart contracts. Besides, the proportion of the common edges between MFG and CCG to the total edges of CCG is very low, 0.06, suggesting that almost all smart contracts do not receive ETH from their creators. A potential reason is that the purpose of contract creators is to earn money from the users of their smart contracts, and thus they do not deposit Ether into their smart contracts. Moreover, the proportion of the common edges between MFG and CIG to the total edges of MFG is very low, 0.08, indicating that almost all Ether transfers do not go to the invoked smart contracts. Such observation is accordant with the results in Section 5.1 and Table 3 that the edges of MFG outnumber the edges of CIG and about one half of smart contracts are not invoked. Additionally, the proportion of the common edges between MFG and CIG to the total edges of CIG is low, 0.2, indicating that the contract callers do not often transfer Ether from/to the called smart contracts. The reason may be that whether the caller transfers Ether from/to the called contract depends on the business logic of the callee. For example, a token contract for transferring tokens always refuses Ether transfer.

Evolution Analysis. The numbers of nodes and edges increase, because more Ether is transferred over time. Figure 8 depicts the evolution of MFG in terms of clustering coefficient, associativity coefficient, Pearson coefficient, number of SCCs, size of the largest SCC, number of WCCs, size of the largest WCC, and $|E(MFG) \cap E(CCG)|$ in order. The first point refers to the data collected in the first month after the launching of Ethereum. The clustering coefficient fluctuates between 0.1 to 0.3, indicating that nodes tend to cluster together at all times. The assortativity coefficient increases from about −0.1 to nearly 0, indicating that the disassortativity of MFG remains neutral over time.

The Pearson coefficient fluctuates between 0.4 to 0.8 except a few points (highlighted by the red box), indicating that an obviously positive correlation maintains over time. After investigating the points that are smaller than 0.4, we find many outliers that lead to small Pearson coefficients, because Pearson coefficient is very sensitive to the presence of outliers [5]. For each outlier, its indegree significantly differs with its outdegree, and hence Pearson coefficient becomes small. Table 11 lists three such outliers. 209c is a smart contract belonging to an exchange market that receives money from many accounts (large indegree) and sends the money to only one account (small outdegree). *257b* is an EOA belonging to an exchange market. It receives money from a few accounts (small indegree) and sends the money to many users of the exchange market (large outdegree). *EA67* is the account of a mining pool for receiving mining reward and sending the reward to miners of the mining pool. Thus, *EA67* has a small indegree and a large outdegree.

Table 11. Three Outliers in Computing Pearson Coefficient of MFG

Outlier	indegree	outdegree	identity	category
209c	158,802	1	Poloniex	exchange
257b	31	117,217	Kraken	exchange
EA67	18	20,514	Ethermine	mining pool

The size of the largest SCC becomes larger over time. A possible explanation is that the largest SCC consists of a big exchange market and its users. Therefore, an increasing size means that more and more users leverage the exchange market to trade ETH/tokens. The number of SCC increases over time, and a possible explanation is that more and more applications have been deployed to Ethereum over time. The size of the largest WCC increases, because the size of Ethereum blockchain increases over time. The number of WCC is relatively stable around 100 with a few outliers, because a WCC contains many SCCs. By investigating these outliers, we find some miners that do not belong to any mining pools, and hence there are many small WCCs containing these miners. After a period of time, these small WCCs merge into large WCCs, because miners send Ether to other accounts or trade Ether in exchange markets.

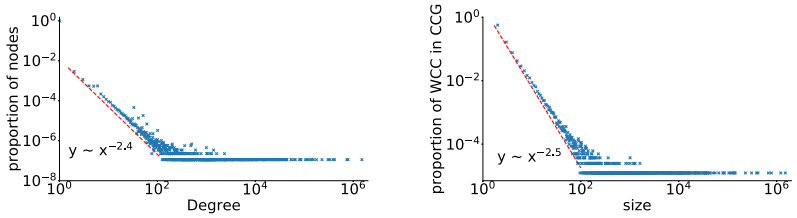
The two proportions $|E(MFG) \cap E(CCG)|/|E(MFG)|$ and $|E(MFG) \cap E(CCG)|/|E(CCG)|$ fluctuate but keep small over time. We find that in the 23rd month, these two proportions reach their largest values in history. By the end of the 23rd month, 806,862 smart contracts are created, and 227,288 (28%) out of them received Ether from their creators. After checking these contract creation behaviors, we find that a smart contract (i.e., *6090*), which is one of the top 10 most important nodes in both MFG (Table 10) and CCG (Table 12), increases the two proportions. *6090* creates 198,897 smart contracts by the end of the 23rd month, and each of them receives Ether from *6090* during contract creation. The reason is due to the business logic of *6090*, as shown in Listing 1. When a user places a new bid, the function newBid() (Line 1) will be invoked, and the bid price should not be lower than *minPrice* (Line 3). Then, *6090* creates a new smart contract for each new bid and deposits the bid price sent from the user into the created smart contract (Line 3). Therefore, the money of the user rather than the money of *6090* is sent to the smart contract serving the user.

```
1 function newBid(bytes32 sealedBid) payable {
2   if(address(sealedBids[msg.sender][sealedBid]) > 0) throw;
3   if(msg.sender < minPrice) throw;
4   Deed newBid = (new Deed).value(msg.value)(msg.sender);
5   sealedBids[msg.sender][sealedBid] = newBid;
6   NewBid(sealedBid, msg.sender, msg.value);
7 }
```

Listing 1. *6090* sends Ether to the smart contracts created by itself.

6.3 CCG Analysis

Degree distribution. Figure 9(a) shows the degree distribution of CCG. We do not present indegree/outdegree distributions, because the indegree of an account is either 0 (EOA) or 1 (smart contract). For the same reason, we do not measure the correlation of indegree and outdegree by computing Pearson coefficient. The degree distribution of CCG follows the power law, meaning that a few nodes create a large number of smart contracts. This observation matches the analysis in Section 5.3 that smart contracts outnumber application developers.



(a) Degree distribution (b) Size of WCC

Fig. 9. Degree distribution and size of WCC of CCG.

Table 12. Top 10 Most Important Nodes of CCG Evaluated by Degree Centrality

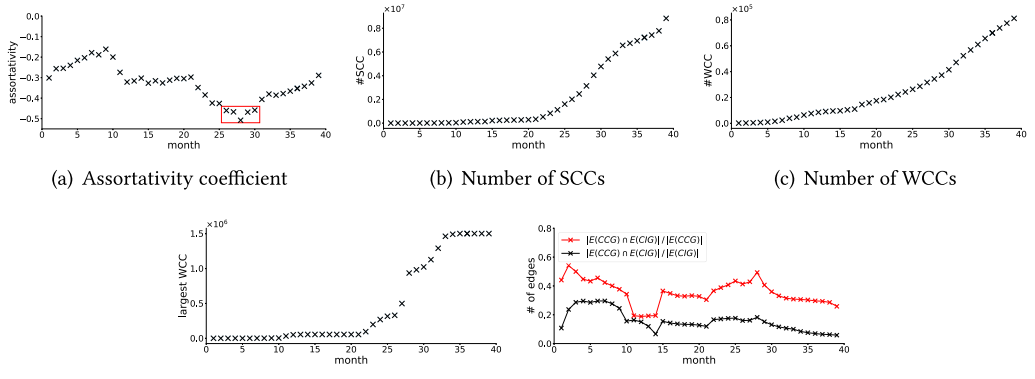
account	a3c1	0000	71d2	6090	b42b	279b	17bc	4f01	5a4e	a998
type	sc	sc	sc	sc	EOA	sc	EOA	sc	sc	sc
degree	1,501,242	780,268	743,617	415,208	388,517	333,423	306,597	229,735	177,461	154,057
identity	Bittrex	Gastoken	/	name service	Poloniex	/	/	/	/	/
category	exchange	token	/	fundamental	exchange	/	/	/	/	/
Ether	10,461,657	0	31,165	3,302,192	5,000,248	110,689	604,997	2,386	0	0
# invoke	8,129,457	3,240	1,928,399	2,303,199	1,392,528	1,316,129	195,662	545,382	1	1
T2(3)×size(*sc_set*)	750,620	390,133	371,808	207,603	194,258	166,711	153,298	114,867	88,730	77028

Clustering coefficient. Table 7 also lists the analysis results of CCG. Its clustering coefficient equals to zero, because if two contracts are created by the third node, they cannot create each other. Indeed, a contract cannot be created twice.

Assortativity coefficient. The assortativity coefficient is −0.29, indicating weak disassortativity of CCG; that is to say, large-degree nodes tend to connect small-degree nodes. That is, if an account creates many contracts, the created smart contracts are unlikely to create many other smart contracts. This observation matches Figure 2(b) that only a few smart contracts create smart contracts. And, we observe that the huge number of created contracts by the same account are similar and do simple tasks that do not need to create other smart contracts.

SCC/WCC. As expected, the largest SCC has only 1 node, because there are no cycles in CCG. Surprisingly, the size of the largest WCC of CCG is 1,501,271, accounting for 17.2% $(1,501,271/8,725,788)$ of all contracts. The root of this WCC is an EOA (*F892*), which directly or indirectly created 17.2% of the total contracts. In CCG, contract A is said to directly create contract B if there is an edge from A to B. A is said to indirectly create B if A does not directly create B but there is a path from A to B. After inspecting the WCC rooted at *F892*, we reveal that the contract *a3c1*, which is created by *F892*, directly creates 1,501,241 contracts. Unsurprisingly, *a3c1* is the most important node of CCG, as shown in Table 12, because it creates many contracts. After deanonymization by the method described in Section 7.3, we find that all nodes of the WCC belong to Bittrex, an exchange market. Figure 9(b) shows that most WCCs are very small, indicating that most applications just consist of a few of smart contracts. Particularly, the number of WCCs whose sizes are larger than 10 is 5,554, accounting for 7% $(5,554/81,177)$ of all WCCs.

Degree centrality. Since the indegree of an account is either 0 or 1, PageRank cannot provide informative values, and thus we just evaluate the importance of nodes in CCG by degree centrality. Table 12 lists the top 10 most important nodes in CCG. The last three rows contain information for anomaly detection in Section 7.2. The EOA *b42b* belongs to an exchange market, Poloniex. The smart contract *0000* is a token that is detailed in Section 7.2.



(a) Assortativity coefficient (b) Number of SCCs (c) Number of WCCs

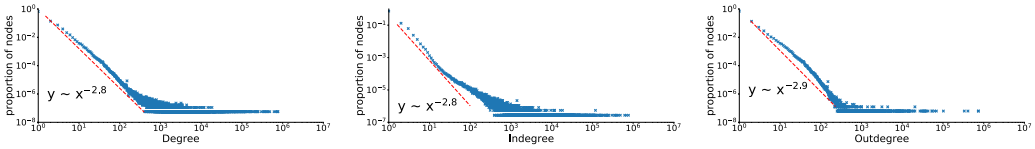
(d) Size of the largest WCC (e) $|E(CCG) \cap E(CIG)|$

Fig. 10. Trend of CCG.

Common edges. The proportion of the common edges between CCG and CIG to the total edges of CCG is low, 0.26, indicating that 74% of smart contracts are not invoked by their creators. Besides, the proportion of the common edges between CCG and CIG to the total edges of CIG is very low, 0.08, suggesting that most contract invocations are not from contract creators to the smart contracts created by them. A reasonable explanation for the two proportions is that smart contracts aim to serve other users, rather than their creators.

Evolution Analysis. The numbers of nodes and edges increase, because more smart contracts are deployed on the blockchain over time. The clustering coefficient remains 0 at all time (i.e., CCG cannot contain triangles), because if two contracts A and B are created by C, then A cannot create B and vice versa. The size of the largest SCC is still 1, because there is no cycle in CCG. Figure 10 presents the evolution of CCG in terms of assortativity coefficient, number of SCCs, number of WCCs, the size of the largest WCC, and $|E(CCG) \cap E(CIG)|$ in order.

The assortativity coefficient fluctuates between -0.4 and -0.1 except for a few outliers (highlighted by the red box) that are smaller than -0.4, indicating that CCG is a disassortative graph at all times. By investigating these outliers, we find that some accounts create a large number of contracts; however, these created contracts do not create new contracts. Therefore, there are lots of edges that connect large-degree nodes to small-degree nodes, leading to a very small assortativity coefficient. For example, two of the top 10 most important nodes (Table 12), a3c1 and 71d2, create 547,123 and 478,482 contracts in 28 months, respectively. However, these 1,025,605 contracts do not create new contracts. Note that the number of created contracts shown in Table 12 is the number accumulated in 39 months. The number of SCCs increases, because new contracts are deployed over time. The number of WCCs increases, because the developers of smart contracts increase over time, considering the root of a WCC is a developer. The size of the largest WCC increases, indicating that there is a developer who creates lots of contracts. The root of the largest WCC is *F892*, which belongs to an exchange market Bittrex, and therefore all the contracts in the WCC are created by Bittrex. The two proportions $|E(CCG) \cap E(CIG)|/|E(CCG)|$ and $|E(CCG) \cap E(CIG)|/|E(CIG)|$ fluctuate but keep small over time. $|E(CCG) \cap E(CIG)|/|E(CCG)|$ reaches its historical low value, 0.18 at the 12th month. By the end of the 12th month, 111,487 smart contracts are created, and 90,511 (82%) out of them are not invoked by their creators. By investigating these 90,511 smart contracts, we find that a smart contract 8b3b obviously impacts the two proportions, because it creates 48,776 smart contracts, but none of them are invoked by the end of the 12th month.



(a) Degree (b) Indegree (c) Outdegree

Fig. 11. Degree/Indegree/Outdegree distributions of CIG.

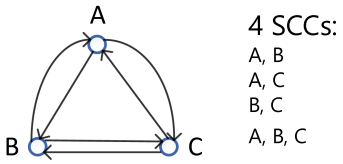


Fig. 12. A community structure with 3 nodes and 4 SCCs.

Insight 4. A small number of developers created lots of smart contracts. By downloading and inspecting all smart contracts (2,061,393) created by EOAs, we find that only 159,127 (i.e., 7.7%) are unique.

6.4 CIG Analysis

Degree distribution. Figure 11 gives the degree/indegree/outdegree distributions of CIG, all of which follow the power law. Indegree distribution reveals that the majority of contracts are invoked by a few accounts, and outdegree distribution indicates that most accounts invoke a few contracts. We can learn that not all contracts are widely used and similarly not all users frequently use Ethereum.

Pearson coefficient. Table 7 gives the measurement results of CIG. We do not consider EOAs when evaluating the correlation of indegree and outdegree (i.e., Pearson coefficient), because the indegree of an EOA is always zero. Pearson coefficient is 0.01, indicating a very weak correlation between indegree and outdegree [27]. An explanation is that whether a smart contract invokes another smart contract is determined by the program logic of the contract rather than whether the smart contract is invoked.

Clustering coefficient. The clustering coefficient approaches 0, meaning that if an account A calls contract B and C, then B and C are very unlikely to call each other. One possible reason is that B and C are independent modules where interactions between them are rare.

Assortativity coefficient. The assortativity coefficient approaches 0, so there is no obvious relation between the degree of a node A and the degree of a node B, which connects to A. The observation is reasonable, because how nodes connect with each other in CIG is determined by the program logic of smart contracts rather than the degree of nodes.

SCC/WCC. The number of SCCs (17,744,593) is larger than the number of smart contracts (8,725,788), indicating that there should be many community structures where nodes intensely connect with each other. Figure 12 presents such a community structure that has three nodes and four SCCs. The largest SCC contains 103,367 nodes. After inspecting the SCC, we find that it contains two popular applications. One is *dd9f*, which is one of the top 10 most important nodes of CIG (Table 13, detailed below). The other one is *6090*, one of the top 10 most important nodes of CCG (Table 12). The number of WCCs is significantly fewer than the number of SCCs, and the largest

Table 13. Top 10 Most Important Nodes of CIG Evaluated by PageRank

account	a3c1	74fd	2bd2	86fa	f230	209c	7da8	d261	fa52	dd9f
type	sc	sc	sc	sc	sc	sc	sc	sc	sc	sc
PR	0.022	0.016	0.012	0.011	0.0072	0.007	0.0069	0.0067	0.006	0.0057
identity	Bittrex	SiaCashCoin	AmIOnTheFork	EOSToken Contract	TronToken	Poloniex	Golem_MultiSig	OmiseGo	Kraken	LastWinner
category	exchange	token	attack-related	token	token	exchange	wallet	token	exchange	gamble

Table 14. Top 10 Most Important Nodes of CIG Evaluated by Degree Centrality

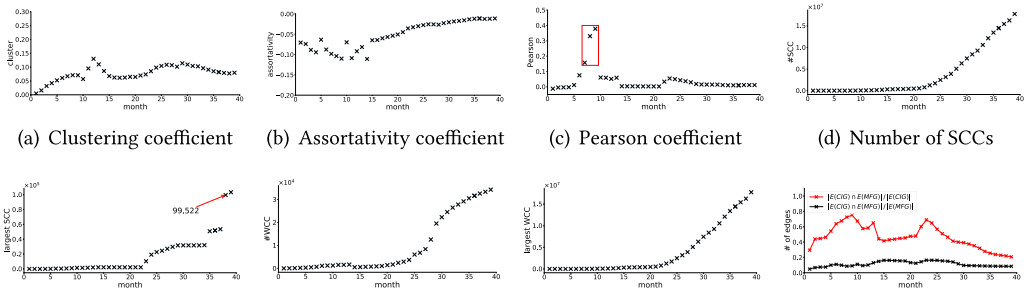
account	86fa	71d2	fbb1	74fd	a3c1	f230	6090	1f04	8a88	d261
type	sc	sc	sc	sc	sc	sc	sc	sc	sc	sc
degree	768,641	743,622	734,563	680,654	620,426	483,538	441,776	430,229	403,045	376,657
identity	EOSToken Contract	/	Bittrex	SiaCash Coin	Bittrex	TronToken	name service	NePay	IG	OmiseGo
category	token	/	exchange	token	exchange	token	fundamental	token	token	token

WCC is much larger than the largest SCC. The explanation is that a WCC can contain many SCCs. For example, if an EOA calls a contract in an SCC, and it also calls a contract in another SCC, the two SCCs will be merged into one WCC. Note that they cannot be merged into one SCC, because for each node in the two SCCs, we cannot find a path from the node to the EOA (EOA cannot be called).

PageRank. Table 13 lists the top 10 most important nodes ranked by PageRank in CIG. We have several observations from this table. First, all top 10 most important nodes are smart contracts. This observation is reasonable, because smart contracts are the most important nodes in contract invocation graph. The contract *2bd2* is an important node in CIG, because it is used to prevent the attacks that replay transactions between the old chain (i.e., Ethereum) and new forked chain (i.e., Ethereum Classical) [25]. Four out of 10 most important nodes are token contracts that realize four different cryptocurrencies based on Ethereum. The contract *7da8* is a wallet contract that increases security by requiring multiple parties to agree on transactions before execution [22]. That is, transactions can be executed only when confirmed by a predefined number of owners [22]. Therefore, 8 out of the 10 most important nodes are financial applications. The contract *dd9f*, named LastWinner, is a gambling application. LastWinner is so popular that Ethereum was congested by the high volume of transactions invoking it [37].

Degree centrality. Table 14 lists the top 10 most important nodes ranked by degree centrality in CIG. The columns with grey background indicate that these accounts also belong to the top 10 most important nodes of CIG evaluated by PageRank. All 10 accounts in this table are also smart contracts. Six out of 10 accounts are token contracts. Such a finding is expected, because tokens play a critical role in various applications, and thus token behaviors (e.g., transfer some tokens from one account to another account) frequently happen in Ethereum.

Evolution Analysis. The numbers of nodes and edges increase, because more smart contracts are invoked over time. Figure 13 demonstrates the evolution of CIG in terms of clustering coefficient, associativity coefficient, Pearson coefficient, the number of SCCs, the size of the largest SCC, the number of WCCs, the size of the largest WCC, and $|E(MFG) \cap E(CIG)|$ in order. The clustering coefficients in most checkpoints are smaller than 0.1, indicating that contracts do not tend to cooperate at all times. The assortativity coefficient fluctuates between -0.15 and 0, indicating that there is no obvious relation between the degrees of two connected nodes at all times.



(a) Clustering coefficient (b) Assortativity coefficient (c) Pearson coefficient (d) Number of SCCs

(e) Size of the largest SCC (f) Number of WCCs (g) Size of the largest WCC (h) $|E(MFG) \cap E(CIG)|$

Fig. 13. Trend of CIG.

The Pearson coefficients fall into $[0, 0.1]$ except a few outliers (highlighted in the red box), indicating a very weak correlation between indegree and outdegree at all times. After investigating these outliers, we find some smart contracts that have large indegrees and outdegrees, and therefore Pearson coefficient becomes large due to these contracts. For example, *6c8f* is an open-source smart contract providing alarm clock service with a large indegree (1,021) and a large outdegree (1,029), indicating that 1,021 accounts call *6c8f* and *6c8f* calls 1,029 smart contracts. After reading its source code, we find that if a contract A calls the function updateDefaultPayment() of *6c8f*, updateDefaultPayment() will call A again, and hence, the indegree of *6c8f* is close to its outdegree.

The number of SCCs and the number of WCCs increase over time. The reason may be that more applications have been deployed to the blockchain, and more users invoke these applications over time. The size of the largest SCC also increases, and a possible explanation is that a large SCC is formed due to popular applications. For example, the largest SCC in the 38th month (marked in Figure 13(e)) contains 99,522 nodes, which is much bigger than the largest SCC in the 37th month. After investigating the two SCCs, we find that the SCC in the 38th month contains a community structure, in which 35,703 accounts call *dd9f* (the gambling application, LastWinner), and *dd9f* calls 35,695 smart contracts, but the SCC in the 37th month does not contain such a community structure. The size of the largest WCC increases, because the EOAs of this WCC glue many SCCs. More specifically, the largest WCC contains 17,748,461 nodes, and 13,212,413 (74%) of them are EOAs.

$|E(MFG) \cap E(CIG)|/E(MFG)|$ fluctuates but keeps small over time, but $|E(MFG) \cap E(CIG)|/E(CIG)|$ has some peaks. We find that a few hot applications can obviously increase the common edges between MFG and CIG. For example, $|E(MFG) \cap E(CIG)|/E(CIG)|$ reaches its historical high value, 0.69 at the 23rd month, when 1,833,418 edges in CIG are invoked and 1,265,838 also appear in MFG. After inspecting these 1,265,838 edges, we find the account *6090* contributes 15% $(184,020/1,265,838)$ of $|E(MFG) \cap E(CIG)|/E(CIG)|$. Please recall that *6090* is one of the 10 most important nodes in MFG (Table 10), CCG (Table 12), and CIG (Table 14). Listing 2 shows that when a user invokes the function unsealBid() of *6090*, it will call bid.value() (Line 5), bid.setBalance() (Line 6), bid.creationDate() (Line 8), and bidcloseDeed() (Line 9) in the smart contract bid created by itself.

Insight 5. Financial applications, such as exchange markets and tokens, dominate Ethereum, because they are the most important nodes in money transfer (Table 9), contract creation (Table 12), and contract invocation (Table 13), although Ethereum allows different types of applications [3].

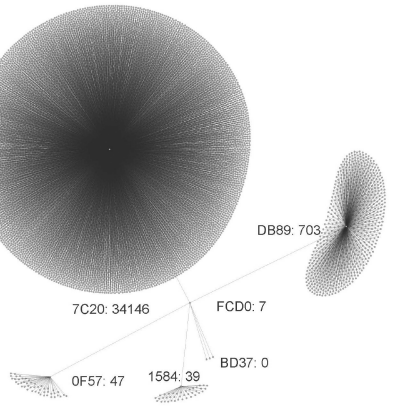


Fig. 14. Attack forensics of *BD37*.

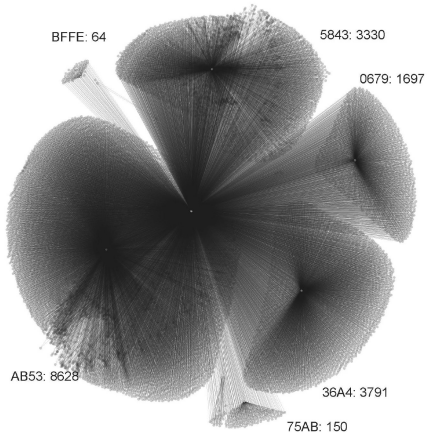


Fig. 15. Attack forensics of *5843*.

```
function unsealBid(bytes32 _hash, uint _value, bytes32 _salt) {
......
  Deed bid = sealedBids[msg.sender][seal];
......
  uint value = min(_value, bid.value());
  bid.setBalance(value, true);
......
  } else if(...|| bid.creationDate() ...) {
    bid.closeDeed(995);
......
  }
}
```

Listing 2. *6090* invokes the smart contracts created by itself.

7 APPLICATIONS BASED ON GRAPH ANALYSIS

Besides inspecting individual graphs, we propose new approaches based on the three graphs to address three important security issues in Ethereum, including attack forensics (Section 7.1), anomaly detection (Section 7.2), and deanonymization (Section 7.3).

7.1 Attack Forensics

Given a malicious smart contract, attack forensics intends to find all accounts controlled by the attacker. To achieve this goal, we correlate CCG and CIG to obtain all smart contracts created by the attacker and all accounts invoking such smart contracts. More precisely, we first compute the WCC containing the malicious contract from CCG to collect all contracts (directly or indirectly) created by the root. Then, for each node in the WCC, we locate all callers from CIG. If a caller is a smart contract, then we backtrack in CIG until reaching an EOA. Eventually, all nodes in the WCC and all nodes (directly or indirectly) invoking the nodes in the WCC should be controlled by the attacker.

Figure 14 shows our analysis result of a real case, where the node *BD37* is a malicious contract for a DoS attack [9]. We randomly select 10K contracts created by *7C20* for the ease of drawing this

ALGORITHM 1: Detection of abnormal contract creation

Inputs: x, the detected account
MFG, money flow graph
CCG/CIG, contract creation/invocation graphs
T_1, T_2, T_3, thresholds
Outputs: True/False, x is abnormal/benign

```
1   sc_set = created_sc(CCG, x);
2   if size(sc_set) < T1 return False;
3   for each node y in sc_set
4       caller_set = inedge(CIG, y);
5       for each edge z in caller_set
6         num + = z.weight;
7       sender_set = inedge(MFG, y);
8       for each edge s in sender_set
9         value + = s.weight;
10  if num > T2 × size(sc_set) || value > T3 × size(sc_set)
11      return False;
12  else return True;
```

figure. The WCC containing *BD37* roots in the node *FCD0*. The notation x:y indicates that node x creates y contracts. We can see that the WCC has 34,942 nodes, and *7C20* creates 34,146 contracts among them. 34,939 contracts in the WCC are invoked, and there are 43 callers that invoke the contracts in the WCC. Five out of 43 callers belong to the WCC, and the other 38 callers are EOAs. Therefore, attack forensics show that to launch the attack, the attacker creates 34,941 contracts and leverages 43 EOAs to call 34,939 created contracts. Figure 15 shows our analysis result of another real case, where the node *5843* is a malicious contract for stealing ETH [38]. The WCC contains 17,704 nodes whose root is *8761*. 395 smart contracts in the WCC are invoked, and 10 callers invoke these 395 contracts. None of the callers belong to the WCC, and 8 out of them are EOAs. Hence, our conclusion is that to launch the attack, the attacker creates 17,703 contracts and leverages 10 EOAs to call 395 created contracts.

7.2 Anomaly Detection

We design a new approach to detect abnormal contract creation, which consumes lots of resources (e.g., disk, network) by creating a great number of unused contracts, because every Ethereum client has to maintain a copy of the blockchain. An intuitive detection approach is to count the number of created contracts. Unfortunately, it is not accurate, because benign applications (e.g., exchange markets) may also create many contracts for their businesses (Table 12). As shown in Algorithm 1, our detection algorithm regards an account as abnormal if it creates lots of contracts that are *rarely used* to transfer money and invoked, because such accounts will waste computing resources of the blockchain. This algorithm correlates the three graphs to detect abnormal activities.

The inputs of the detection algorithm include an account x, MFG, CCG, CIG, and three thresholds (i.e., $T1$, $T2$, $T3$). It returns True if x launches a campaign of abnormal smart contract creation, or False otherwise. It first obtains all contracts directly or indirectly created by x from CCG (Line 1). If the number is smaller than $T1$, then the algorithm considers x to be benign (Line 2), because not many accounts are created. For each created smart contract y (Line 3), all edges pointing to it are obtained from CIG (Line 4). Since the weight of an edge of CIG is the number of invocations, *num* is the total number of invocations to all contracts belonging to the WCC (Line 6). Besides, the amount of Ether (i.e., *value*) transferred by y is computed based on MFG (Line 9). If *num* is smaller

than $T2$×size(*sc_set*) and *value* is smaller than $T3$×size(*sc_set*), then x is considered as abnormal, since the contracts in the WCC are rarely used both in money transfer and contract invocation.

By setting $T1 = 10{,}000$, $T2 = T3 = 0.5$, we apply our anomaly detector to all 48,298,522 accounts and reveal 48 abnormal accounts that created many unused contracts. Note that the thresholds could be learned from the activities of normal accounts, and we will investigate it in future work. Only one abnormal account (*0000*) is open-source, which creates 774,235 contracts directly or indirectly. Interestingly, 3 (*0000*, *5a4e*, *a998*) of the top 10 most important nodes of CCG (Table 12) are abnormal accounts. Table 12 presents the amount of transferred Ether (row 6) and the number of invocations (row 7) of all smart contracts created by the top 10 most important nodes of CCG. The results of $T2(3)$×size(*sc_set*) are also given in the last row.

Abnormal contract creation can result from various reasons, e.g., Denial of Service (DoS) attacks for consuming resources, attacks for stealing tokens, business logic of smart contracts. We propose a method to recognize DoS attacks automatically, but the investigation of other reasons needs manual efforts to inspect the program logic of the smart contracts' bytecode (e.g., what smart contracts attempt to do?). Our automated approach is based on the following observation: The smart contracts produced by DoS attacks have no functionalities; instead, the created smart contracts are just for wasting computing resources (e.g., disk resources for storing smart contracts, network resources for synchronizing smart contracts). A smart contract with functionalities should contain a jump-table-like structure that directs contract invocations to the called functions [49]. Our approach considers an abnormal account as an account for launching DoS attacks if at least one of its created smart contracts does not have such jump tables. Our approach finds 71% (34/48) of abnormal accounts that launch DoS attacks.

We then manually investigate 6 out of the remaining 14 abnormal accounts to discover the reasons for abnormal contract creation. *0000* is Gastoken, which aims to save money for users [26]. More precisely, the account who creates a contract should pay an execution fee, because contract creation consumes computing resources of all Ethereum clients. The execution fee is computed by the multiplication of gas cost and gas price, where gas cost evaluates how many units of gas will be consumed and gas price is evaluated in Ether of one gas unit [3]. Ethereum encourages developers to self-destruct smart contracts by refunding some units of gas, because self-destruction will release the disk space for storing contracts [3]. Therefore, Gastoken suggests users to purchase tokens with a low gas price, and Gastoken will create many contracts during purchasing. Then users can earn Ether by burning tokens with a high gas price, because these created contracts will be self-destructed during the burning process. However, we observe that users purchase many tokens without burning, thus lots of created contracts are not self-destructed. A possible explanation is that users are bullish of Gastoken so they tend to hold the tokens rather than burn these tokens. Therefore, the account *0000* creates a huge number of unused smart contracts due to its business logic. The account *470f* creates *0000*, and thus it is also abnormal.

175e creates *5a4e* and *a998*, and the latter two create 177,460 and 154,056 contracts, respectively. Most of these contracts are unused, and hence all three accounts are abnormal. When *5a4e* creates contracts, each created contract invokes a token contract, named SiaCashCoin; that is, the construction function of each created contract calls SiaCashCoin. SiaCashCoin rewards the accounts who invoke SiaCashCoin with some tokens for promotion. SiaCashCoin ensures that each account can get reward only once. Therefore, *5a4e* can steal tokens (i.e., get reward many times, which is unexpected by SiaCashCoin) by creating many contracts, and each created contract gets reward once. *a998* behaves similarly as *5a4e*, which steals flamingostar token. *2207* behaves like *175e*, which steals various tokens, such as SiaCashCoin, Bankcoin Cash. Therefore, the four abnormal accounts steal tokens.

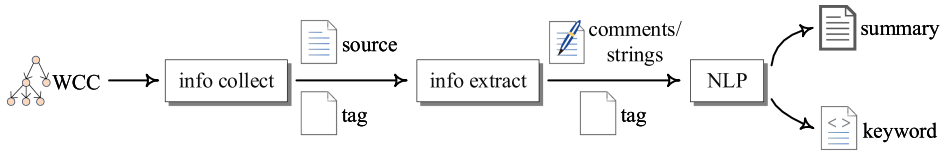


Fig. 16. Procedures of deanonymization.

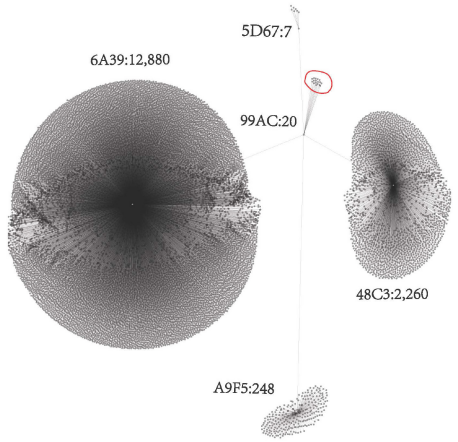


Fig. 17. The WCC rooted in *99AC*.

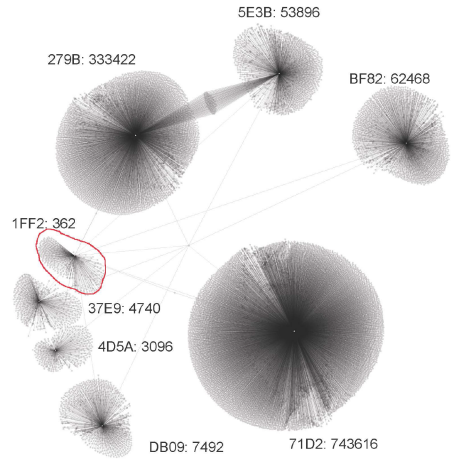


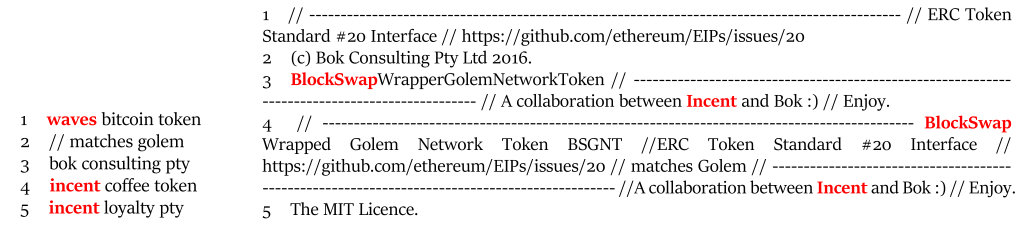
Fig. 18. The WCC rooted in *1Ff2*.

7.3 Deanonymization

Deanonymization aims to associate account addresses with real identities, which facilitates fighting against crimes (e.g., money theft, money laundering, blackmail). However, deanonymization of Ethereum is an open problem, because users do not need to provide their identities for using Ethereum. A common practice of deanonymization is inferring the identity of an account by combining multiple sources (e.g., name tag, source code, discussion board) of information.

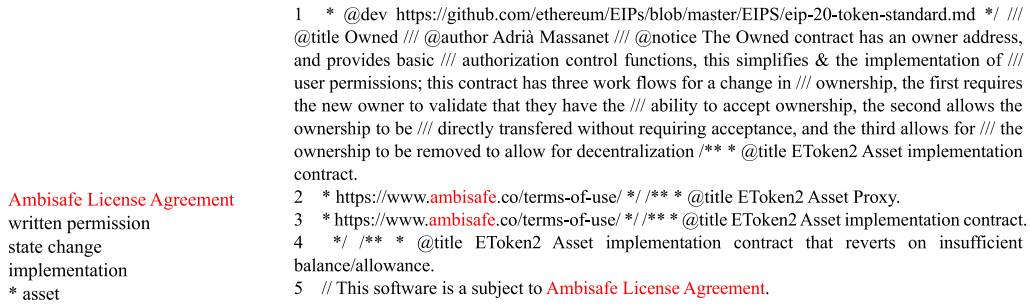
In this study, we propose to infer the identity of a group of accounts leveraging the WCC of CCG. Recall that the root of a WCC is an EOA and the other nodes are smart contracts that are directly or indirectly created by the root. Therefore, all nodes in a WCC should have the same identity (i.e., all accounts of the WCC are created by the same person or organization). One advantage of our approach is that more information can be obtained from all nodes of a WCC than from a single node. Indeed, the more information we have, the higher chance we have to successfully infer the identity. Figure 16 presents the procedures of our deanonymization approach. It accepts a WCC and then collects information (e.g., name tag, source code) about all nodes belonging to the WCC. After that, we extract comments and constant strings from source code and tags, since they may contain hints of identity. Then, we leverage natural language processing (NLP) to distill key information (e.g., keyword, text summary). We plan to leverage more types (e.g., code structures of bytecode) of information to infer identity in future work.

We use a real case to illustrate the procedures: the identity of node *6A39*, because it creates lots of (i.e., 12,880) contracts. Figure 17 illustrates the WCC containing *6A39* whose root is *99AC*. Note that *x*:*y* in this figure means that node *x* creates *y* contracts. Although *6A39* creates many contracts, we do not find useful information from it and all its created contracts. Moreover, we do not find identity information for the root. By traversing the WCC, we find that some contracts (circled in this



(a) keyword (b) sentence

Fig. 19. Keywords and sentences of the WCC rooted in *99AC*.



(a) keyword (b) sentence

Fig. 20. Keywords and sentences of the WCC rooted in *1Ff2*.

figure) have name tags and open source, which are directly created by the root *99AC*. After processing the comments and constant strings by RAKE, an NLP tool (http://textminingonline.com/tag/rake), we extract five keywords and five summarized sentences from all comments, constant strings, and name tags, as shown in Figure 19. From the results, we can easily infer that those contracts are developed by a corporation, named Incent Loyalty Pty, and are used to swap tokens on Wave blockchain. We confirm the analysis results by visiting Incent's website. Consequently, we reveal the identity of all 15,416 accounts belonging to the WCC via graph-based deanonymization.

As another example, we present how to infer the identity of node *71d2*, the No. 3 most important node of CCG, by leveraging CCG. Unfortunately, we do not find any identity information from *71d2* and all 743,616 smart contracts created by it. Therefore, we try to infer its identity by investigating the WCC containing *71d2* whose root is *1Ff2*, as shown in Figure 18. For the ease of drawing this figure, we randomly select 20K nodes from the WCC. The WCC contains 1,210,377 accounts and 32 (inside the circle) out of them are open-source smart contracts that are all created by *1Ff2*. We then use RAKE to extract five keywords and five summarized sentences from their source code. From the results, as shown in Figure 20, we can easily discover that all 1,210,377 accounts of the WCC belong to a company named Ambisafe.

8 RELATED WORK

This section retrospects graph analysis of Ethereum. Before introducing related studies, we first discuss the differences between Ethereum with Bitcoin, which make it inappropriate to directly apply the techniques for, and the insights from the graph analysis of Bitcoin to Ethereum.

Table 15. Comparison of Ethereum with Bitcoin

Blockchain	Account	Balance	Change	Multi-inputs	Multi-outputs	Many addresses
Ethereum	√	√	×	×	×	×
Bitcoin	×	×	√	√	√	√

8.1 Ethereum vs. Bitcoin

Although Bitcoin has a basic scripting mechanism, it does not support complex programs like smart contracts, because it does not provide complex program structures (e.g., loops) [1]. New techniques based on cryptographic protocols (e.g., RSK [31], a sidechain) aim at enhancing Bitcoin with the capability of running complex programs and are still in active development. Therefore, we focus on the money transfer capability of Bitcoin hereinafter. Table 15 lists the major differences of Ethereum with Bitcoin in terms of money transfer. Different from Ethereum, there are *no* accounts or balances in Bitcoin. The basic block of a Bitcoin transaction is an *unspent transaction output* (UTXO) [42]. When a user receives BTC (the native cryptocurrency of Bitcoin), the amount is recorded in the blockchain as a UTXO. Besides, a user's BTC may be scattered in multiple UTXOs, and the concept of an account does not exist in Bitcoin [6]. The balance of a user is calculated by the wallet application, which scans the blockchain and aggregates all UTXO belonging to that user [6]. By contrast, each account of Ethereum has a unique address and a *balance* field to record the money [42].

A transaction of Bitcoin can have *change*, because after a UTXO is created, the UTXO cannot be cut in half [6]. If a UTXO is larger than desired, then the whole UTXO will be consumed and *change* will be produced in that transaction. In Ethereum, an account sends an exact amount of money to another, and hence there is no change. A transaction of Bitcoin can have multiple inputs and multiple outputs, because the wallet can aggregate multiple UTXOs belonging to the same user (i.e., multiple inputs) for payment and send money to many recipients in one transaction (i.e., multiple outputs). Note that a transaction of Ethereum comes from one sender to one receipt, and thus a transaction of Ethereum has just one input and one output. Moreover, a user of Bitcoin often has many addresses, because the client of Bitcoin generates a new address to receive the change of a UTXO [8], but in Ethereum one EOA has a unique address. Therefore, the techniques for, and the insights from the graph analysis of Bitcoin cannot directly apply to Ethereum due to their many differences.

8.2 Graph Analysis of Ethereum

This article extends our previous conference paper [18], and the major extensions are presented in Section 1. Kiffer et al. records contract creation and contract invocation by client instrumentation to compute many statistics of Ethereum (e.g., how many contracts are created by contracts) [30]. However, they do not record the complete contract invocation data. For example, they lack the invocation from EOAs to contracts. They define a graph, named *smart contract topology*, where each node is a contract and each edge indicates contract invocation [30]. But, they do not characterize the topology using graph analysis. Charlier et al. construct a graph, named *smart contract graph*, where each node is a contract and each edge indicates Ether transfer between two nodes [14]. They study some metrics (e.g., degree distribution) of the graph [14]. However, their paper does not explain how to collect relevant data for constructing the graph. Besides, they just study a very small graph that consists of hundreds of nodes.

Chan and Olmsted store the entities (e.g., accounts, transactions) and the relations (e.g., Ether transfer) among entities into a graph database for fast querying [13]. But, they do not conduct

graph analysis on the recorded data. Somin et al. investigate a special kind of contract invocation, named *token transfer* [41]. A token is a cryptocurrency realized in smart contracts, and token transfer means that an account transfers some amount of tokens to another account. They construct a graph by parsing contract invocation, where each node is an account and each edge indicates token transfer between two accounts [41]. They find that the graph follows power law [41]. Cachin et al. propose a transaction graph, named TDAG, as a general semantics for modeling various blockchains, including Ethereum, Bitcoin, and Hyperledger Fabric [12]. However, they neither study the characteristics of TDAG nor develop applications on it.

9 CONCLUSION

We conduct the *first* systematic study to characterize Ethereum via graph analysis. By instrumenting an EVM client, we collect all transactions and then construct three graphs (i.e., MFG, CCG, and CIG) to characterize the activities of Ether transfer, contract creation, and invocation, respectively. By analyzing these graphs through various metrics, we obtain many new observations and insights, which help people to have a deep understanding of Ethereum. Moreover, we propose new approaches that leverage those graphs to address three security issues in Ethereum, and the evaluation through real cases demonstrates their effectiveness.

In the future, we plan to extend our work from the following aspects. First, we will conduct a thorough study of Ethereum using more graph metrics. Second, we plan to develop more graph-based applications based on MFG, CCG, and CIG. Third, we will try to set the threshold parameters used for anomaly detection automatically by leveraging machine learning. Fourth, we plan to incorporate the code structure of smart contracts to improve deanonymization. Finally, since the information required for deanonymization may not be written in English, we plan to apply multilingual NLP in the future.

REFERENCES

[1] Bitcoin Wiki. 2018. Bitcoin Script. Retrieved from https://en.bitcoin.it/wiki/Script.
[2] Coin Market Cap. 2018. CrypoCurrency Market Capitalizations. Retrieved from https://coinmarketcap.com/.
[3] Ethereum. 2018. Ethereum Homestead Documentation. Retrieved from http://www.ethdocs.org/en/latest/.
[4] Ethereum community. 2018. Etherscan, The Ethereum Blockchain Explorer. Retrieved from https://etherscan.io/.
[5] Mokhtar Bin Abdullah. 1990. On a robust correlation coefficient. *J. Roy. Stat. Soc. Series D (The Stat.)* 39, 4 (1990), 455–460.
[6] Andreas M. Antonopoulos. 2014. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media, Inc.
[7] Annika Baumann, Benjamin Fabian, and Matthias Lischke. 2014. Exploring the Bitcoin network. In *Proceedings of the International Conference on Web Information Systems and Technologies*.
[8] Bitcoin. 2017. Change. Retrieved from https://en.bitcoin.it/wiki/Change.
[9] Bok. 2016. Ethereum Network Attackers IP Address Is Traceable. Retrieved from https://www.bokconsulting.com.au/blog/ethereum-network-attackers-ip-address-is-traceable/.
[10] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.* 30 (1998), 107–117.
[11] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. 2000. Graph structure in the web. *Comput. Netw.* 33 (2000), 309–320.
[12] Christian Cachin, A. D. Caro, Pedro Moreno-Sanchez, Björn Tackmann, and Marko Vukolic. 2017. The transaction graph for modeling blockchain semantics. Retrieved from https://eprint.iacr.org/2017/1070.pdf.
[13] Wren Chan and Aspen Olmsted. 2017. Ethereum transaction graph analysis. In *Proceedings of the International Conference for Internet Technology and Secured Transactions*.
[14] Jérémy Charlier, Sofiane Lagraa, and Jerome Francois. 2017. Profiling smart contracts interactions with tensor decomposition and graph mining. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.
[15] Ting Chen, Xiaoqi Li, Xiapu Luo, and Xiaosong Zhang. 2017. Under-optimized smart contracts devour your money. In *Proceedings of the IEEE 24th International Conference on Software Analysis, Evolution and Reengineering*.

[16] Ting Chen, Xiaoqi Li, Ying Wang, Jiachi Chen, Zihao Li, Xiapu Luo, Man Ho Au, and Xiaosong Zhang. 2017. An adaptive gas cost mechanism for Ethereum to defend against under-priced DoS attacks. In *Proceedings of the International Conference on Information Security Practice and Experience*.

[17] Ting Chen, Zihao Li, Hao Zhou, Jiachi Chen, Xiapu Luo, Xiaoqi Li, and Xiaosong Zhang. 2018. Towards saving money in using smart contracts. In *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results*.

[18] Ting Chen, Yuxiao Zhu, Zihao Li, Jiachi Chen, Xiaoqi Li, Xiapu Luo, Xiaodong Lin, and Xiaosong Zhang. 2018. Understanding Ethereum via graph analysis. In *Proceedings of the IEEE Conference on Computer Communications*.

[19] Aaron Clauset, Cosma Rohilla Shalizi, and Mark E. J. Newman. 2009. Power-law distributions in empirical data. *SIAM Rev.* 51 (2009), 661–703. Issue 4.

[20] Ethereum. 2018. JavaScript API. Retrieved from https://github.com/ethereum/wiki/wiki/JavaScript-API#web3ethgettransaction.

[21] Etherscan. 2018. Ethereum Developer APIs. Retrieved from https://etherscan.io/apis#accounts.

[22] Gnosis. 2018. Ethereum Multisignature Wallet. Retrieved from https://github.com/Gnosis/MultiSigWallet.

[23] Alex Greaves and Benjamin Au. 2015. Using the bitcoin transaction graph to predict the price of bitcoin. Retrieved from http://snap.stanford.edu/class/cs224w-2015/projects_2015/Using_the_Bitcoin_Transaction_Graph_to_Predict_the_Price_of_Bitcoin.pdf.

[24] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. 2018. MadMax: Surviving out-of-gas conditions in Ethereum smart contracts. In *Proceedings of the ACM on Programming Languages*.

[25] Hiddentao. 2016. Prevent replay attacks. Retrieved from https://github.com/hiddentao/smart-solidity-docs/blob/master/PreventReplayAttacks.md.

[26] IC3. 2016. GasToken.io—Cheaper Ethereum Transactions, Today. Retrieved from https://gastoken.io/.

[27] Phanny ITH. 2009. Guideline for Interpreting Correlation Coefficient. Retrieved from https://www.slideshare.net/phannithrupp/guideline-for-interpreting-correlation-coefficient.

[28] Jerome Kehrli. 2016. Blockchain 2.0—From Bitcoin Transactions to Smart Contract applications. Retrieved from https://www.niceideas.ch/blockchain_2.0.pdf.

[29] Kentstate. 2019. SPSS Tutorials: Pearson Correlation. Retrieved from https://libguides.library.kent.edu/SPSS/PearsonCorr.

[30] Lucianna Kiffer, Dave Levin, and Alan Mislove. 2018. Analyzing Ethereum's contract topology. In *Proceedings of the Internet Measurement Conference*.

[31] Sergio Demian Lerner. 2015. RSK-White Paper Overview. Retrieved from https://docs.rsk.co/RSK_White_Paper-Overview.pdf.

[32] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. 2017. A survey on the security of blockchain systems. *Fut. Gen. Comput. Syst.* (Aug. 2017).

[33] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*.

[34] Damiano Di Francesco Maesa and Laura Ricci. 2016. An analysis of the Bitcoin users graph: Inferring unusual behaviours. In *Proceedings of the International Workshop on Complex Networks and their Applications*.

[35] Natarajan Meghanathan. 2016. Assortativity analysis of real-world network graphs based on centrality metrics. *Comput. Inf. Sci.* 9 (2016), 7–25. Issue 3.

[36] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2013. A fistful of bitcoins: Characterizing payments among men with no names. In *Proceedings of the Internet Measurement Conference*.

[37] Nick Mwenda. 2018. Last winner game jams Ethereum network—Is Ethereum (ETH) dominance at stake? Retrieved from https://cryptoglobalist.com/2018/08/17/last-winner-game-jams-ethereum-network-is-ethereum-eth-dominance-at-stake/.

[38] PeckShield. 2018. Pwning Fomo3D Revealed: Iterative, Pre-Calculated Contract Creation For Airdrop Prizes! Retrieved from https://medium.com/@peckshield/pwning-fomo3d-revealed-iterative-pre-calculated-contract-creation-for-airdrop-prizes-31944a01387e.

[39] Fergal Reid and Martin Harrigan. 2011. An analysis of anonymity in the Bitcoin system. In *Security and Privacy in Social Networks*. 197–223.

[40] Sara Nadiv Soffer and Alexei Vazquez. 2005. Network clustering coefficient without degree-correlation biases. *Phys. Rev. E* 71, 5 (2005).

[41] Shahar Somin, Goren Gordon, and Yaniv Altshuler. 2018. Network analysis of ERC20 tokens trading on Ethereum blockchain. In *Proceedings of the International Conference on Complex Systems*.

[42] Flora Sun. 2018. UTXO vs Account/Balance Model. Retrieved from https://medium.com/@sunflora98/utxo-vs-account-balance-model-5e6470f4e0cf.

[43] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin Vechev. 2018. Securify: Practical security analysis of smart contracts. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*.

[44] Fabian Vogelsteller and Vitalik Buterin. 2015. ERC-20 Token Standard. Retrieved from https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md.

[45] Xiaomin Wang, Matthieu Latapy, and Michele Soria. 2012. Deciding on the type of the degree distribution of a graph from traceroute-like measurements. *Int. J. Comput. Netw. Commun.* 4, 3 (2012), 151–168.

[46] Gavin Wood. 2018. Ethereum: A secure decentralised generalised transaction ledger. Retrieved from https://ethereum.github.io/yellowpaper/paper.pdf.

[47] Steve Y. Yang and Jinhyoung Kim. 2015. Bitcoin market return and volatility forecasting using transaction network flow properties. In *Proceedings of the IEEE Symposium Series on Computational Intelligence*.

[48] Chen Zhao and Yong Guan. 2015. A Graph-based investigation of bitcoin transactions. In *Proceedings of the IFIP International Conference on Digital Forensics*.

[49] Yi Zhou, Deepak Kumar, Surya Bakshi, Joshua Mason, Andrew Miller, and Michael Bailey. 2018. Erays: Reverse engineering Ethereum's opaque smart contracts. In *Proceedings of the USENIX Security Symposium*.

Received June 2019; revised November 2019; accepted January 2020