Understanding Ethereum via Graph Analysis

Ting Chen*†, Yuxiao Zhu‡§, Zihao Li*, Jiachi Chen†, Xiaoqi Li†, Xiapu Luo†¶, Xiaodong Lin‖, Xiaosong Zhang*
*Center for Cybersecurity, University of Electronic Science and Technology of China, China
†Department of Computing, The Hong Kong Polytechnic University, China
‡School of Management, Guangdong University of Technology, China
§School of Big Data and Strategy, Guangdong University of Technology, China
‖Department of Physics & Computer Science, Wilfrid Laurier University, Canada
Email: brokendragon@uestc.edu.cn, zhuyuxiao.mail@gmail.com, gforiq@qq.com,
{csxqli,csjcchen,csxluo}@comp.polyu.edu.hk, xlin@wlu.ca, johnsonzxs@uestc.edu.cn

***Abstract*— Being the largest blockchain with the capability of running smart contracts, Ethereum has attracted wide attention and its market capitalization has reached 20 billion USD. Ethereum not only supports its cryptocurrency named Ether but also provides a decentralized platform to execute smart contracts in the Ethereum virtual machine. Although Ether's price is approaching 200 USD and nearly 600K smart contracts have been deployed to Ethereum, little is known about the characteristics of its users, smart contracts, and the relationships among them. To fill in the gap, in this paper, 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 design a new approach to collect all transaction data, construct three graphs from the data to characterize major activities, and discover new observations and insights from these graphs. Moreover, we propose new approaches based on cross-graph analysis to address two security issues in Ethereum. The evaluation through real cases demonstrates the effectiveness of our new approaches.**

I. Introduction

With around 20 billion USD market capitalization, Ethereum is the largest blockchain that supports smart contracts, which are autonomous computer programs that, once started, execute automatically and mandatorily according to the program logic defined beforehand [1]. Two years after its debut in July 2015, there are already around 4M external owned accounts, which are usually normal users or developers, and 600K smart contracts in Ethereum. There are studies about Ethereum's security [2], [3] and performance [4]. However, little is know 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 help us better understand the Ethereum ecosystem.

In this paper, 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. These activities empower users to send money to others, developers to deploy their smart contracts to 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 happened on Ethereum and extracting useful data from them.

¶ The corresponding author.

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 customized Ethereum client that will record the internal transactions (Section IV).

Based on all transaction data, we construct MFG, CCG, CIG to represent money flow, smart contract creation, and smart contract invocation, respectively (Section V). 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 etc. (Section VI). Such investigation leads to new observations and insights. For example, users prefer transferring money rather than calling smart contracts on Ethereum. Moreover, smart contracts for financial applications dominate the Ethereum ecosystem but most smart contracts are not widely used. Besides examining individual graphs, we propose new approaches based on cross-graph analysis to address two security issues in Ethereum (Section VII), including attack forensics for finding accounts controlled by the attacker, and anomaly detection for discovering potential attacks through smart contracts. The experimental result demonstrates the effectiveness of our new approaches. Although some recent works studied Bitcoin through graph analysis [5]–[8], 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 VIII-A.

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 transactions data, and then construct money flow graph (MFG), smart contract creation

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

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 conducting various graph analysis on MFG, CCG and CIG. They help us obtain a better understanding of the Ethereum ecosystem. To foster the research on this topic, we will release the customized Ethereum client, collected data, and processing scripts after paper publication.

(3) We propose new approaches based on cross-graph analysis to handle two security issues in Ethereum, including attack forensics and anomaly detection. The evaluation through real cases shows the effectiveness of new approaches.

The rest of the paper is organized as follows. Section II introduces the background knowledge of Ethereum. After giving an overview of our analysis procedure in Section III, we detail the data collection, graph construction and analysis in Section IV, Section V and Section VI, respectively. Section VII presents the new approaches based on cross-graph analysis for handling two security issues. After reviewing related studies in Section VIII, we conclude the paper in Section IX.

II. Background

Ethereum is the largest blockchain platform that supports smart contracts [9] and its cryptocurrency named Ether. Ether, whose price is approaching 200 USD, can be traded on cryptocurrency exchanges or used to pay for transaction fees and computational services. Developers can create various applications on Ethereum by constructing and deploying smart contracts, which will be executed in the Ethereum virtual machine (EVM). Normal users can transfer Ether to others and use those applications by invoking smart contracts. Ethereum requires users to pay transaction fees for protecting it from frivolous or malicious tasks that will exhaust the resources.

The basic unit in Ethereum is the *account* [9]. There are two kinds of accounts, namely external owned accounts and smart contracts. The major difference between them is that smart contracts contain executable code whereas external owned accounts do not have it. Developers usually prepare smart contracts in high-level languages (e.g., Solidity) and compile them into EVM bytecode. To deploy a smart contract to Ethereum, the creator sends a transaction, whose *data* field contains the bytecode, to the recipient address. Note that a smart contract can be created by either an external owned account or another smart contract. After successful deployment, any user of Ethereum can invoke the smart contract by sending a transaction whose recipient is the contract. The *data* field of that transaction specifies the function in the smart contract to be invoked as well as function's parameters.

Each transaction contains several basic fields. The *recipient* field gives the address of the transaction recipient. If the transaction is used for contract creation, the recipient address is zero. The *signature* field specifies the transaction sender. The *value* field indicates the amount of money transferred from the sender to the recipient. The *data* field contains the bytecode of a smart contract or the information (e.g., parameters) to invoke a smart contract. We focus on transactions because the three major activities of Ethereum, including money transfer, smart contract creation and smart contract invocation are triggered by transactions. The execution of transactions may fail due to many reasons (e.g., invalid bytecode). Once a transaction fails, the effect resulting from applying the transaction will roll back. Hence, when investigating smart contract creation, we only consider successful transactions.

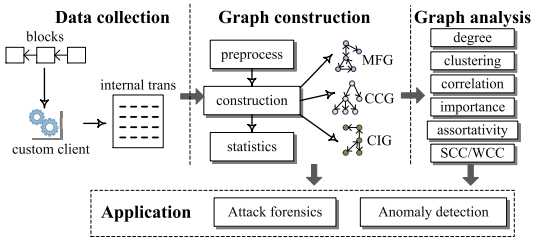


Fig. 1. An overview of our approach.

III. Methodology

As shown in Fig.1, our methodology consists of three phases, which are detailed in the following sections. The first phase, *data collection* (Section IV), collects *all* transaction data for subsequent graph construction. Although the blockchain and all external transactions are publicly available, the internal transactions are not stored in the blockchain. We propose a new approach to obtain internal transactions by modifying the Ethereum client to add instrumentation code. Then, both external and internal transactions are fed to the second phase, *graph construction* (Section V), to construct three graphs, namely 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 last phase, *graph analysis* (Section VI), 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. Based on the statistics and metrics of these graphs, we discover new observations and insights. Besides inspecting individual graphs, we propose new approaches based on cross-graph analysis to handle security issues in Ethereum (Section VII), including attack forensics and anomaly detection.

IV. Data Collection

We collect all transactions (28,502,131 external transactions and 19,759,821 internal transactions) from the launch of Ethereum on July 30th, 2015 to June 10th, 2017. Since external transactions are initialized 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 [10].

However, internal transactions result from the execution of smart contracts, and they are not stored in the blockchain. We 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. One possible approach for obtaining internal transactions is to crawl them from some Ethereum explorer websites (e.g., Etherscan, https://etherscan.io/) maintained by Ethereum community. However, to keep the websites available to all visitors, they usually restrict the amount of data that can

be crawled. Besides, to the best of our knowledge, there are no available tools for collecting internal transactions.

To address this issue, we propose collecting all internal transactions by replaying external transactions in our customized EVM. The rationale behind our solution is that internal transactions are incurred by the execution of smart contracts, which are triggered by external transactions. More precisely, we modify go-ethereum V 1.6.0, a popular Ethereum client implemented in the GO language by inserting the instrumentation code into EVM, a stack-based virtual machine for executing smart contracts.

Ethereum defines 130 operations for EVM [10], and the bytecode of a smart contract can be considered as a sequence of such operations. Ethereum runs smart contracts in EVM, which is in charge of interpreting EVM operations. EVM provides 61 handlers to interpret 61 operations, individually, and 4 special handlers to execute $PUSHx$, $1 \leq x \leq 32$, $DUPx$, $1 \leq x \leq 16$, $SWAPx$, $1 \leq x \leq 16$, and $LOGx$, $0 \leq x \leq 4$, respectively. $PUSHx$ pushes one item on stack, x is the size of item in byte; $DUPx$ duplicates the x-th stack item; $SWAPx$ exchanges the 1st and the $(x+1)$-th stack items; $LOGx$ appends log record with x topics.

By inspecting EVM, we find that five operations can lead to internal transactions, including *CREATE*, *CALL*, *CALLCODE*, *DELEGATECALL* and *SELFDESTRUCT* (an alias of the original *SUICIDE* operation). *CREATE* and *CALL* create and invoke a smart contract, respectively. *CALLCODE* and *DELEGATECALL* also invoke a smart contract, but the callee runs in caller's context. By using them, a smart contract can be loaded as a library by another one. *SELFDESTRUCT* removes the smart contract from the blockchain and sends the remaining money in the smart contract to a designated target. Therefore, we modify the handlers of these five operations.

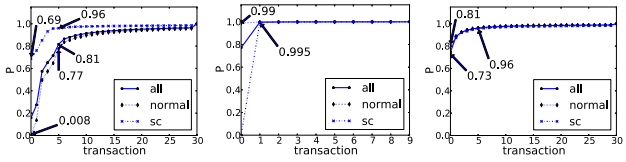
More precisely, in the handler of *CREATE*, we insert the instrumentation code after the success of contract creation to record the address of contract creator, address of the created contract, the amount of Ether deposited in created contract by contract creator. Similarly, in the handlers of *CALL*, *CALLCODE*, *DELEGATECALL*, we log the addresses of caller, callee and the amount of Ether transferred from caller to callee. In the handler of *SELFDESTRUCT*, we record the address of destructed contract, the address to receive the remaining Ether, and the amount of Ether.

V. 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. In the *preprocess* stage, we exclude four types of transactions that are not related to the aforementioned activities. A transaction of the first type sends Ether from an EOA to another but the amount is zero. A transaction of the second type self-destructs a smart contracts and the smart contract has no Ether remaining. Consequently, such transaction does not result in money transfer. The third type of transactions are unsuccessful transactions among EOAs because they do not lead to money transfer. The fourth type of transactions are unsuccessful transactions for smart contract creation since they fail to create contracts. Note that we do not exclude the unsuccessful transactions for contract invocation, because before rolling back the contract will be (partially) executed. Moreover, the transactions for smart contract creation are not considered when building CIG, even if they will execute the construction function of the created smart contract, because according to Ethereum's protocol [10] the code of the construction function will be discarded after smart contract creation and thus no users can invoke it again.

TABLE I
Number of transactions involved in money flow (BG_{MF}), contract creation (BG_{CC}), and contract invocation (BG_{CI})

BG_{MF}	BG_{CC}	BG_{CI}
27,535,903	599,934	9,125,860



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

Fig. 2. Cumulative distribution of the number of transactions

We collect account addresses by parsing the external transactions (i.e., *signature* field and *recipient* field) and the logs of internal transactions. After preprocessing, 2,721,080 accounts (i.e., 2,121,146 s and 599,934 smart contracts) are obtained.

In the *construction* stage, we divide the remaining transactions into three groups (i.e., BG_{MF}, BG_{CC} and BG_{CI}), corresponding to money transfer, smart contract creation, and smart contract invocation. 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 created smart contract and therefore the transaction should be included in both BG_{MF} and BG_{CC}. As another example, when a transaction calls a smart contract, it can send Ether to the contract. In this case, such transaction should be included in BG_{MF} and BG_{CI}. Table I lists the number of transactions in three groups. We can see that the size of BG_{CC} (599,934) is equal to the number of smart contracts, because a smart contract can only be created once. Moreover, users transfer money more frequently ($27,535,903$ vs. $9,125,860$) than interacting with applications through calling smart contracts on Ethereum.

Fig.2 illustrates the distribution of the number of transactions related to the accounts in BG_{MF}, BG_{CC} and BG_{CI}, individually. The solid line, dotted line and dotted 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. Fig.2(a) shows that 0.8% (the point (0, 0.008)) of EOAs do not transfer Ether. In other words, if one EOA is a user, then almost all users transfer (i.e., receive or send) money on Ethereum. However, more than 2/3 smart contracts (the point (0, 0.69)) do not transfer Ether. Moreover, about 81% of accounts (96% of smart contracts and 77% of EOAs) are involved in no more than 5 transactions. That is, most accounts (especially smart contracts) are infrequent in transferring money.

Fig.2(b) shows that 99% of EOAs (point (0, 0.99)) do not create contracts. If an EOA that creates smart contracts is a smart contract developer, the proportion of developers is just 1% of total users. The point (1, 0.995) informs that

TABLE II
STATISTICS OF GRAPHS

graph	# edges	# isolated *EOA*	# isolated *sc*
MFG	5,267,627	16,845	412,293
CCG	599,934	2,098,922	0
CIG	1,281,500	1,554,301	483,404

99.5% of smart contracts are involved in only 1 transaction (i.e., the transaction for contract creation). Hence, we can 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.

Fig.2(c) demonstrates that 73% (point (0, 0.73)) of EOAs do not invoke smart contracts and 81% (point (0, 0.81)) of smart contracts are not invoked. Moreover, about 96% of EOAs call smart contracts no more than 5 times (point (5, 0.96)). Hence, we can learn that not all users use smart contracts frequently. Since 69% of smart contracts do not transfer money, we investigate whether they are invoked. Results show that 60% $(360,341/599,934)$ of smart contracts neither transfer money nor be 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.

A. MFG Construction

Definition V.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. $w : E \to \mathbb{R}_+$ associates each edge with a weight, which is the total amount of transferred Ether along the edge by one or more transactions. Hence, MFG is a weighted directed graph.

We use the terms *account* and *node* interchangeably in the remainder of this paper. Since all nodes have been enumerated in the *preprocess* stage, we add edges to build MFG. Specifically, we parse the *value* field in each external transaction of BG_{MF}, which indicates the amount of money transferred. If there is no edge from the sender to the recipient, we add one and set edge's weight to *value*. If an edge already exists, we increase its weight by *value*. An internal transaction is processed in a similar way. Table II 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 16,845 EOAs (compared to 2,121,146 in total) do not transfer money since they are isolated in MFG. The data in Table II matches the observations from Fig.2(a) that almost all EOAs transfer money whereas about 2/3 smart contracts do not do it.

B. CCG Construction

Definition V.2. $CCG = (V, E)$, where V is a set of nodes, the same as those in Def. V.1. E is a set of edges. $E = \{(v_i, v_j)|v_i \in V, v_j \in V_{sc}\}$, where an edge (v_i, 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. Second, if $(v_i, v_j) \in E$, then $(v_k, v_j) \notin E, \forall k \neq i$. It means that a smart contract cannot be created twice. CCG is actually 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 II) match the observations from Fig.2(b) that the vast majority of EOAs are isolated, i.e., they do not create contracts. Moreover, the smart contracts (599,934) obviously outnumber the EOAs (i.e., $22,224 = 2,121,146 - 2,098,922$), which create contracts. If an EOA which creates smart contracts is a developer, 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.

C. CIG Construction

Definition V.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 account v_i invokes the contract 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.

To construct CIG, we process every (external and internal) transaction in BG_{CI} to extract the addresses of transaction sender and recipient. Then, wes 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 II shows the statistics of CIG, which match the observations from Fig.2 that 73% $(1,554,301/2,121,146)$ of EOAs do not call smart contracts, and 81% $(483,404/599,934)$ of smart contracts are not invoked.

We obtain the following insights by building the graphs.

Insight 1. Users prefer to transferring 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.

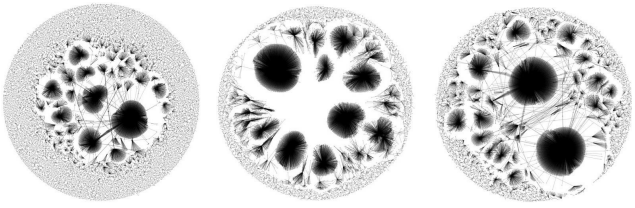
Insight 2. The smart contracts are not widely used. One possible reason is that as shown in Section VI-D there are only a few applications supported by smart contracts and most of them are for financial applications.

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

Fig.3 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 VI.

VI. 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.



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

Fig. 3. Visualization of MFG, CCG and CIG. For the ease of illustration, we randomly select 20,000 edges from each graph to draw the figure.

The degree of a node is the number of edges connecting to the node. 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 summation of the number of contracts invoking it and the number of contracts it invokes.

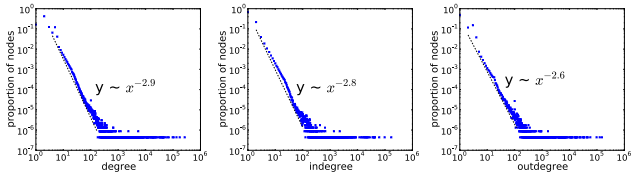
The indegree of a node in a directed graph is the number of edges whose heads end that node. Specifically, the indegree of an account in MFG is the number of accounts sending money to it. The indegree of an EOA in CCG and CIG is 0, since it cannot be created or invoked by other accounts according to the definitions of CCG and CIG (Def. V.2, Def. V.3). The indegree of a contract in CCG is 1, because it should be created only 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 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.

Before explaining the strong/weak connected component (SCC/WCC), we define a *path* in a graph as a sequence of edges that connect a sequence of nodes. A SCC of a directed graph G is the maximal 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. A WCC of a directed graph G is a maximal 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. That is, the direction of edges is ignored when looking for WCC.

Besides, we compute the global clustering coefficient [11] to evaluate the extent to which nodes in a graph tend to cluster together. For example, Section VI-D shows that the clustering coefficient of CIG approaches zero, meaning that the collaboration of contracts in CIG is uncommon. Moreover, we use Pearson coefficient [12] to evaluate the correlation between the indegree and the outdegree of nodes, compute the assortativity coefficient to study the preference for nodes to attach to others, and evaluate node's importance using the PageRank algorithm [13].

A. Inferring Node Identity

It is difficult to discover an account's identity because its address instead of the identity is enough for an account to use Ethereum. We make every effort to reveal an account's identity by exploiting information from many sources. More precisely, since Ethereum allows users to associate name tags with accounts, we could obtain the identity from its name tags. If a smart contract's source code is available, we could infer the identity from it (e.g., comments, keywords). Moreover, we also resort to discussion boards, forums, and search engines because many people may interact with or be interested in the important accounts.



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

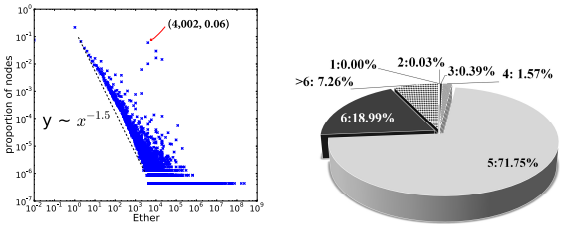
Fig. 4. Degree/Indegree/Outdegree distributions of MFG

Furthermore, we propose a new method that exploits the WCC of CCG to infer the identity of a group of accounts. It exploits the fact that all nodes in a WCC have the same identity because the root of a WCC is an external owned account and the other nodes are smart contracts created by the root directly or indirectly. Hence, our method checks all nodes in the WCC to look for hints about their identity. In the following example, for the ease of presentation, we use the first two bytes of an account's address (the length of an address is 20 bytes in Ethereum) to denote it, and list the original addresses in https://github.com/brokendragon/Ethereum_Graph_Analysis. For the node *6A39* that creates 12,880 contracts, we cannot find useful information from it to infer its identity. Instead, by traversing the WCC containing *6A39*, we notice that some contracts (e.g., *13FC*) created by the root (i.e., *99AC*) of WCC are open source. By investigating the comments, constant strings in the source code, we reveal that all 15,416 accounts in this WCC were created by the corporation, Incent Loyalty Pty.

B. MFG Analysis

Fig.4 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 IV) that a few exchange markets have large degree. The small degree nodes may be the individuals trading with exchange markets.

We then investigate the distribution of Ether, as shown in Fig.5(a). Each point (x, y) in this figure indicates that there are y of total accounts, and each transfers (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 Fig.5(a). Taking it and Fig.4 into consideration, we learn that the outliers are small-degree accounts (i.e., they trade with a few accounts) transferring a lot of money. Fig.5(b) shows the degree statistics of the accounts in an outlier, the point at (4,002, 0.06). 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% of accounts is no larger than 6.



(a) Ether distribution (b) An outlier (4,002, 0.06)

Fig. 5. Ether distribution and statistics of an outlier

TABLE III
METRICS OF THE THREE GRAPHS

graph	cluster	assortativity	Pearson	#SCC	largest SCC	#WCC	largest WCC
MFG	0.17	−0.12	0.44	466,095	1,822,192	81	2,291,707
CCG	0	−0.35	/	622,158	1	22,260	126,246
CIG	0.004	−0.2	0.11	682,984	84	4,088	668,891

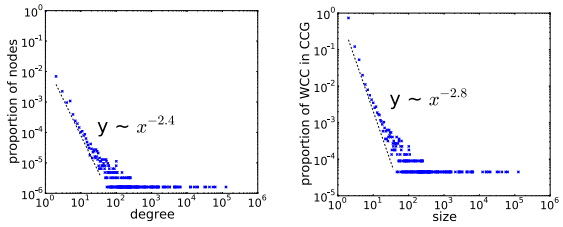
Table III shows the metrics of MFG. Columns 2-8 list the values of the clustering coefficient, assortativity coefficient, Pearson coefficient, number of SCC, size (i.e., how many nodes) of the largest SCC, number of WCC, size of the largest WCC, respectively. The clustering coefficient is large (i.e., 0.17), revealing that if two accounts *A*, *B* trade with a third account *C*, *A* and *B* are likely to trade with each other. The assortativity coefficient is negative, indicating that a large degree account prefers to trade with a small degree account rather than a large degree one. Given the analysis of important nodes in MFG (Table IV), a possible reason is that an exchange market (large degree node) tends to serve many individuals (small degree nodes) rather than other exchange markets.

Pearson coefficient is 0.44 (a moderately strong correlation [14]), 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. The size of the largest SCC is huge, which contains about 86% of nodes $(1,822,192/2,121,146)$. 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., 466,095) in MFG is far more than that of WCCs (i.e., 81). In this case, a WCC may usually 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 next step.

Table IV lists top 10 most important nodes in MFG, ranked by the PageRank algorithm. In this table, *n* means external owned account, *sc* indicates smart contract, *PR* denotes PageRank value. Since some accounts' identities cannot be successfully recovered, we mark their identity and category using '/' in Table IV.

The exchange markets play a key role in money transfer (8 out of 10-accounts). Since the markets tend to serve ordinary users (small degree nodes) rather than other markets (large-degree nodes), the assortativity coefficient is negative. Moreover, exchange markets, which are usually hub nodes connecting to other nodes bidirectionally, result in the huge SCC. The smart contract, *ReplaySafeSplit*, is important because it is used to prevent the attacks that replay transactions between the old chain and new forked chain [15]. Although using this smart contract is optional, the analysis result indicates that it is used frequently, indicating that the risk of replay attack has received significant attention.



(a) Degree distribution (b) Size of WCC

Fig. 6. Degree distribution and size of WCC of CCG

C. CCG Analysis

Fig.6(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.

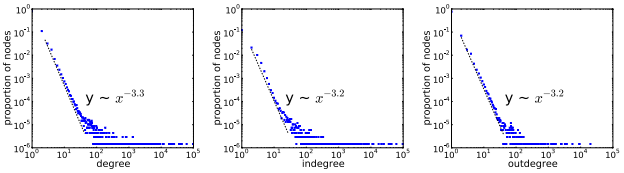
The degree distribution of CCG follows the power law, meaning that a few nodes create a large number of smart contracts. This observation matches the analysis in Section V-B that smart contracts outnumber application developers. Table III also lists the analysis results of CCG. Its clustering coefficient 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. It matches our expectation that the size of largest SCC is 1, because there are no cycles in CCG. Its assortativity coefficient is negative, indicating that large-degree nodes tend to connect small-degree nodes. That is, if an account creates lots of contracts, the created smart contracts are unlikely to create many other smart contracts. This observation matches the analysis in Section V that smart contracts rarely create smart contracts.

Surprisingly, the size of the largest WCC of CCG is 126,246, which accounts for 21% $(126,246/599,934)$ of the total smart contracts. The root of this WCC is an external owned account (i.e., *FDB3*), which directly or indirectly created 21% of the total smart contracts. In CCG, smart contract *A* is said to directly create smart 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*. Fig.6(b) shows that most WCCs are very small. Particularly, the number of WCCs whose sizes are larger than 100 is only 163, accounting for 0.7% $(163/22,260)$ of total WCCs.

Since the indegree of an account is either 0 or 1, the PageRank algorithm does not provide informative values, and thus we use outdegree to evaluate the importance of nodes in CCG. Table V lists the top 10 most important nodes in CCG. The last three rows contain information for anomaly detection in Section VII-B. The node (i.e., *6090*) is a smart contract created by the root of the WCC (i.e., *FDB3*). It is a name service, through which a user can interact with others by specifying the recipient's name rather than its address.

TABLE IV
TOP 10 MOST IMPORTANT NODES OF MFG

account	70FA	209C	FA52	AA1A	1C39	E94B	9E63	96FC	517C	9BCB
type	n	sc	sc	sc	sc	sc	n	n	n	sc
PR	0.057	0.032	0.015	0.015	0.013	0.013	0.012	0.01	0.009	0.008
identity	ShapeShift	Poloniex	Kraken	ReplaySafeSplit	ShapeShift	Bittrex	ShapeShift	Changelly	/	ShapeShift
category	exchange	exchange	exchange	attack-related	exchange	exchange	exchange	exchange	/	exchange



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

Fig. 7. Degree/Indegree/Outdegree distributions of CIG

Moreover, we notice that all smart contracts created by *6090* are open source and the same as *6090*. In other words, *6090* makes lots of copies. The smart contract *EDCE* is created by a leading boutique consulting group, New Alchemy, and the smart contract *6A39* is created by Incent Corp. that supports money transfer between Ethereum and Wave.

Attacks on Ethereum can be detected by inspecting the activities of contract creation. For example, the two attacking accounts *7C20* [16] and *3898* [17] created lots of junk contracts that cost considerable disk space and slow down the synchronization of blockchain. We will detail how to discover abnormal activities similar to the aforementioned DoS attacks in Section VII-B. Without counting the contracts created by attacks and abnormal accounts, we find that about 62% $(296,691/476,746)$ of smart contracts are created by six accounts. Moreover, 5 out of 6 most important applications (without counting attacking and abnormal accounts) are for finance, including exchange markets. Hence, we have the following insight.

Insight 4. A small number of developers created lots of smart contracts. By downloading and inspecting all smart contracts (284,948) created by EOAs, we find that there are only 17,509 (i.e., 6%) unique smart contracts.

D. CIG Analysis

Fig.7 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. Table III 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 0.11, indicating a very weak correlation between indegree and outdegree [14]. The clustering coefficient approaches zero, meaning that if an account *A* calls contract *B* and *C*, then *B* and *C* are very unlikely to call each other. The possible reason is that most contracts are not so complicated that they do not need to invoke others. The negative assortativity coefficient suggests that the large-degree nodes tend to connect small-degree nodes.

Table VI lists top 10 most important nodes ranked by the PageRank algorithm in CIG. Contract *AA1A* is an important node in MFG since it aims at preventing replay attacks. All except two smart contracts in the list are exchange markets. Specifically, TheDAO is a digital decentralized autonomous organization. REG-Augur is an application of Augur, which is a prediction market platform that rewards participants for correctly predicting future real-world events. As a result, we have the following insight.

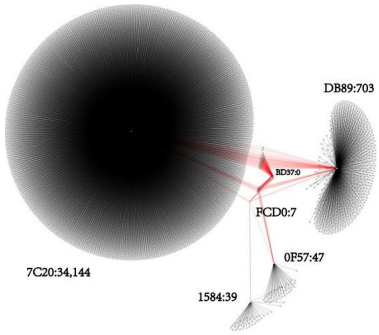


Fig. 8. Attack forensics of *BD37*. We randomly select 10,000 contracts created by *7C20* for the ease of illustration. No contracts created by *7C20* are invoked.

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

VII. APPLICATIONS BASED ON CROSS-GRAPH ANALYSIS

Besides inspecting individual graphs, we propose new approaches based on cross-graph analysis to address two important security issues in Ethereum, including attack forensics (Section VII-A) and anomaly detection (Section VII-B).

A. 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, 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 are controlled by the attacker.

Fig.8 shows our analysis result of a real case, where the node *BD37* is a malicious contract for a DoS attack [16]. The WCC containing *BD37* roots in the node *FCD0*. The notation *x*:*y* indicates that node *x* creates *y* contracts. We can see that *7C20* creates 34,144 contracts, which is one of the top important nodes in CCG (Table V). Fig.8 highlights the nodes in red if they are invoked and the edges connecting to the red nodes. By doing so, it presents attacking-related nodes

TABLE V
TOP 10 MOST IMPORTANT NODE OF CCG

account	6090	B42B	8B3B	42DA	7C20	EDCE	0536	29DF	3898	6A39
type	sc	n	sc	n	sc	sc	n	sc	n	sc
outdegree	126,233	62,786	54,445	43,134	34,144	31,453	20,219	18,605	15,994	12,880
identity	name service	Poloniex	/	YUNBI	/	New Alchemy	Poloniex	/	/	Incent
category	fundamental	exchange	anomaly	exchange	attack	finance	exchange	anomaly	attack	finance
Ether	4,753,280	1,049,504	144	15,392,521	0	4505	890	0	0	64,827
# invoke	3	33,577	10	1,218,859	2	42,025	15,481	2	1,122	2,699
$T2(3)\times$size(sc_set)	63,117	31,394	27,224	21,568	17,074	15,728	10110	9,304	7,997	6,440

TABLE VI
TOP 10 MOST IMPORTANT NODES OF CIG

account	209C	AA1A	FA52	BB9B	1C39	E94B	A744	9BCB	BFC3	48C8
type	sc	sc	sc	sc	sc	sc	sc	sc	sc	sc
PR	0.071	0.05	0.033	0.029	0.028	0.027	0.016	0.015	0.011	0.011
identity	Poloniex	ReplaySafeSplit	Kraken	TheDAO	ShapeShift	Bittrex	Golem	ShapeShift	Poloniex	REP-Augur
category	exchange	attack-related	exchange	organization	exchange	exchange	exchange	exchange	exchange	predict

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;
```

and edges from both CCG and CIG. Using this approach, we find that the attacker controls 34,983 accounts in total, where 34,939 are smart contracts but only 9 were invoked.

B. 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 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 enough because benign applications (e.g., exchange markets) may also create many contracts for their businesses (Table V). As show in Algorithm 1, our detection algorithm regards an account as abnormal if it creates lots of contracts that are rarely used to transfer money or invoked. This algorithm correlated 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 $T1$, 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 $T2\times$size (sc_set) and *value* is smaller than $T3\times$size (sc_set), x is considered as abnormal since the contracts in the WCC are rarely used both in money transfer and contract invocation.

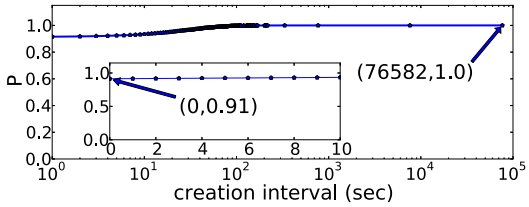


Fig. 9. Cumulative distribution of time interval between two consecutive contract creation events

TABLE VII
COMPARISON OF ETHEREUM WITH BITCOIN

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

Table V presents the amount of transferred Ether and the number of invocations of all smart contracts created by the top 10 most important nodes of CCG. The results of $T2(3)\times$size(sc_set) are also given in Table V. By setting $T1$, $T2$ and $T3$ to be 10,000, 0.5, 0.5 respectively, we detect four abnormal accounts, *7C20*, *3898*, *8B3B* and *29DF*. Note that the thresholds could be learned from the activities of normal accounts, and we will investigate it in future work. For *7C20*, both the amount of transferred Ether (i.e., 0 Ether) and the number of invocations (i.e., 2) are very small, and hence it is regarded as abnormal by our detection algorithm. As another example, *3898* creates lots of contracts in short time period, which could slow down the propagation speed of blocks significantly. Fig.9 shows the CDF of the time intervals between two consecutive contract creation events due to *3898*. We can see that 91% (point (0, 0.91)) of contracts are created in a very short period of time. This can serve as another indicator for detecting abnormal activities. Actually, *7C20* and *3898* have already been confirmed to be attacks [16], [17]. The other two suspicious accounts have similar activities, and we have reported them to the Ethereum community.

VIII. RELATED WORK

To the best of our knowledge, this is the *first* investigation on Ethereum through graph analysis. Although there are some graph-based studies on Bitcoin, our investigation of Ethereum needs new analysis methods due to the differences in functionalities and protocols between Ethereum and Bitcoin.

A. Ethereum vs. Bitcoin

Although Bitcoin has a basic scripting mechanism, it does not support complex programs like smart contracts. Moreover, Table VII lists the major differences in money transfer. There are *no* accounts or balances in Bitcoin but Ethereum has them. The basic block of a Bitcoin transaction is an *unspent transaction output* (UTXO). When a user receives BTC (the cryptocurrency used in Bitcoin), the amount is recorded in the blockchain as a UTXO. Thus, a user's BTC may be scattered in multiple UTXOs [18]. The balance of a user is calculated by the wallet application which scans the blockchain and aggregates all UTXO belonging to that user [18]. By contrast, each account of Ethereum associates an address and has a *balance* field to record the money.

A transaction of Bitcoin can have *change*, because after a UTXO is created it cannot be cut in half. If a UTXO is larger than desired, the whole UTXO will be consumed and *change* will be produced in that transaction. In Ethereum, an account sends 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. Moreover, a user of Bitcoin often has many addresses whereas Ethereum associates one EOA with one address.

B. Graph Analysis of Bitcoin

Recent studies [19]–[24] examine the P2P overlay of Bitcoin. Miller et al. find influential nodes in P2P topology [20] while others [21], [22] characterize network latency, client latency etc. Donet et al. investigate propagation time, location distribution, and network stability of Bitcoin P2P network [24]. Bojja et al. leverage P2P topology and P2P traffic for deanonymization [19], [23]. Different with them, we focus on Ethereum's money transfer and smart contracts rather than its communication network. Reid et al. [5] abstracts Bitcoin into a transaction graph, where each node represents a transaction and a directed edge from node A to node B means that the output of A is the input of B. Due to Bitcoin's unique features in Table VII, Meiklejohn et al. propose address clustering to associate users with their addresses [6]. Then, Bitcoin can be modeled as a user graph, based on which some applications have been developed, such as deanonymization [5], and money laundering detection [7], [8]. Ranshous et al. [25] introduce a special graph whose nodes are either transactions or addresses to detect money laundering. Note that these approaches cannot be directly applied to Ethereum due to the obvious differences between Ethereum with Bitcoin.

IX. Conclusion

We conduct the *first* systematic study to characterize Ethereum through graph analysis. By customizing EVM client, we collect all transaction data and then construct three graphs (e.g., MFG, CCG and CIG) to characterize the activities of money transfer, contract creation, and contract invocation, respectively. By analyzing these graphs through various metrics, we obtain many new observations and insights, which help people have a deep understanding of Ethereum. Moreover, we propose new approaches based on cross-graph analysis to address two security issues in Ethereum and the evaluation through real cases demonstrates their effectiveness. In future, we will conduct a more thorough study of Ethereum including its evolution using more graph metrics and develop more applications (e.g., detection of DoS attacks). The customized Ethereum client, collected data, and processing scripts can be found at https://github.com/brokendragon/Ethereum_Graph_Analysis.

Acknowledgment

This work is supported in part by the Hong Kong GRF (PolyU 152279/16E), the HKPolyU Research Grants (G-YBJX), Shenzhen City Science and Technology R&D Fund (No. JCYJ20150630115257892), the Scientific Supporting Plan of Sichuan Province.

References

[1] J. Kehrli. (2016) Blockchain 2.0 – from bitcoin transactions to smart contract applications. [Online]. Available: https://goo.gl/CcDx4J
[2] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems*, Aug. 2017.
[3] T. Chen, X. Li, Y. Wang, J. Chen, Z. Li, X. Luo, M. H. Au, and X. Zhang, "An adaptive gas cost mechanism for ethereum to defend against under-priced dos attacks," in *Proc. ISPEC*, 2017.
[4] T. Chen, X. Li, X. Luo, and et al., "Under-optimized smart contracts devour your money," in *Proc. SANER*, 2017.
[5] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Proc. SocialCom*, 2011.
[6] S. Meiklejohn, M. Pomarole, G. Jordan, and et al., "A fistful of bitcoins: characterizing payments among men with no names," in *Proc. IMC*, 2013.
[7] C. Zhao and Y. Guan, "A graph-based investigation of bitcoin transactions," in *Proc. DigitalForensics*, 2015.
[8] D. Maesa, A. Marino, and L. Ricci, "An analysis of the bitcoin users graph: inferring unusual behaviours," in *Proc. Complex networks*, 2016.
[9] (2017) What is ethereum? [Online]. Available: https://goo.gl/egvz4h
[10] G. Wood. (2014) Ethereum: A secure decentralised generalised transaction ledger. [Online]. Available: https://goo.gl/y6f9eM
[11] W. Just, H. Callender, and M. LaMar. (2015) Clustering coefficients. [Online]. Available: https://goo.gl/TFc4id
[12] L. Rodgers and W. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, Feb. 1988.
[13] A. Langville and C. Meyer, "Deeper inside pagerank," *Internet Mathematics*, Jan. 2004.
[14] (2009) Guideline for interpreting correlation coefficient. [Online]. Available: https://goo.gl/mC9uab
[15] T. Rapp. (2016) How to deal with the ethereum replay attack. [Online]. Available: https://goo.gl/gvAuYY
[16] Bok. (2016) Ethereum network attackers ip address is traceable. [Online]. Available: https://goo.gl/xAVJmg/
[17] latetot. (2016) Attacker is gearing up again for new spam deluge. [Online]. Available: https://goo.gl/F3aS5J
[18] A. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc, 2014.
[19] B. Venkatakrishnan, G. Shaileshh, and P. Viswanath, "Dandelion: Redesigning the bitcoin network for anonymity," in *Proc. POMACS*, 2017.
[20] A. Miller, J. Litton, A. Pachulski, and et al. (2015) Discovering bitcoins public topology and influential nodes. [Online]. Available: https://goo.gl/2KgDwi
[21] D. Christian and R. Wattenhofer, "Information propagation in the bitcoin network," in *Proc. P2P*, 2013.
[22] T. Neudecker, P. Andelfinger, and H. Hartenstein, "Timing analysis for inferring the topology of the bitcoin peer-to-peer network," in *Proc. UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld*, 2016.
[23] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin p2p network," in *Proc. CCS*, 2014.
[24] J. Donet, C. Pérez-Sola, and J. Herrera-Joancomartí, "The bitcoin p2p network," in *Proc. FC*, 2014.
[25] S. Ranshous, A. Cliff, K. Sean, and et al., "Exchange pattern mining in the bitcoin transaction directed hypergraph," in *Proc. FC*, 2017.