Automated labeling of unknown contracts in Ethereum.

Robert Norvill
Department of Electrical Engineering and Computer Science
University of Bradford
R.E.Norvill at Bradford.ac.uk

Beltran Borja Fiz Pontiveros
Sedan Group, SnT
University of Luxembourg
beltran.fiz at uni.lu

Radu State
Sedan Group, SnT
University of Luxembourg
radu.state at uni.lu

Irfan Awan
Department of Electrical Engineering and Computer Science
University of Bradford
I.U.Awan at Bradford.ac.uk

Andrea Cullen
Department of Electrical Engineering and Computer Science
University of Bradford
A.J.Cullen at Bradford.ac.uk

***Abstract*—Smart contracts have recently attracted interest from diverse fields including law and finance. Ethereum in particular has grown rapidly to accommodate an entire ecosystem of contracts which run using its own crypto-currency.**

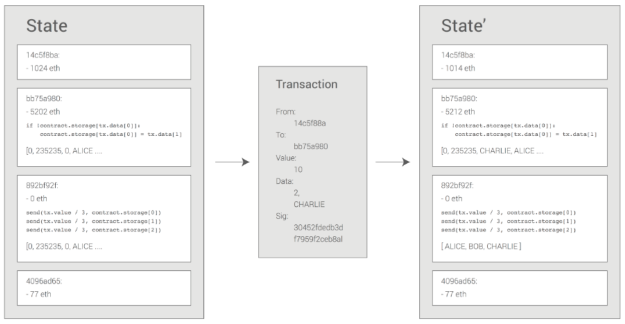
Smart contract developers can opt to verify their contracts so that any user can inspect and audit the code before executing the contract. However, the huge numbers of deployed smart contracts and the lack of supporting tools for the analysis of smart contracts makes it very challenging to get insights into this eco-environment, where code gets executed through transactions performing value transfer of a crypto-currency. We address this problem and report on the use of unsupervised clustering techniques and a seed set of verified contracts, in this work we propose a framework to group together similar contracts within the Ethereum network using only the contracts publicly available compiled code. We report qualitative and quantitative results on a dataset and provide the dataset and project code to the research community.

1. Introduction

Ethereum [1] is a blockchain based cryptocurrency system that aims to provide a decentralized general purpose computer. The programs that run on this decentralized computer are referred to as smart contracts and are automatically enforced through the blockchain validation process that is carried out independently by all full nodes. Full nodes are those that download and validate the whole blockchain. These nodes do not need to trust any other node since they can validate the whole transaction history.

The underlying currency of Ethereum is called ether, and each contract that resides in the blockchain can store ether and bytecode that represents the application. Contracts in Ethereum have a single block of bytecode that is executed by calls, but high level languages like Solidity automatically define a function selector at the beginning of the block that redirects calls to the appropriate part of the bytecode. Users use accounts to execute code in a smart contract by sending messages (i.e. transactions), which include their account address and, as the destination address, the smart contract identifier. If the destination address is not specified, the transaction will result in a new contract being created instead. Unlike accounts, which are managed by users, contracts cannot initiate transactions themselves unless executed by a respective command in the code. In this way, contracts are reactive to transactions sent by the users (i.e. accounts).

The Ethereum blockchain stores the current state of the Ethereum state machine, and the transactions accepted in a block are what moves the state machine to the next state. An example of this state transition can be seen in Figure 1.



```
if !contract.storage[tx.data[0]]:
    contract.storage[tx.data[0]] = tx.data[1]
```

```
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
```

Figure 1: Transitions in Ethereum State machine

In order to execute code in the contract, each execution requires a certain amount of ”gas” which is specified in each transaction together with the rate that defines the price in ether for each unit of gas. The rationale for the usage of gas is to avoid contracts executing code without any special purpose or costs, which avoids a waste of resources and limits the incentives for denial of service attacks. To save storage space Ethereum allows contracts to delete themselves when they are no longer necessary. The community is incentivized to do so by refunding the costs required for creating contracts in the beginning.

While the application scenarios of smart contracts are manifold and are still evolving, many legal aspects are still under investigation. Especially for financial institutions which have a regulatory requirement to monitor account activities for anti-money laundering (AML) purposes. One challenge with AML is that it rarely manifests as the activity of a ”single” person, business, account, or transaction. The detection therefore requires behavioral pattern analysis of transactions occurring over time and involves a set of related real-world entities. Given the anonymity of most accounts within the Ethereum network, the task of AML has become particularly difficult.

Given that only the compiled code is publicly available, it is up to the contract's creator to publicly share their contract so that users can use it. Some contract developers share the source code along with a description of their contract's purpose so that other users can audit the contract, and once they are satisfied with the validity of the code, start using it. This process of verification is optional.

One of the main problems in Ethereum is the discovery of contracts. Unless its creator invests in promoting the contract through specialized forums or dedicated websites, most remain anonymous and without description.

The main motivation behind this work is firstly to see if by using only the compiled code we can discover a set of clusters or communities within the Ethereum smart contract ecosystem, and through automated or manual inspection provide a label to these clusters.

One of the contributions of our work is the compilation of a dataset using publicly available sources of verified Ethereum contracts; this includes the contracts compiled code, source code, and other meta data shared such as contract name and user comments.

In addition we provide a method for automatic labeling of unknown contracts using only the compiled code. This tool could be used by contract creators to monitor similar smart contracts being deployed in the Ethereum marketplace, and by security experts to detect potentially malicious contracts. Automatic labeling of contracts is also useful for building a search engine on the Ethereum marketplace. Smart contracts could be discovered automatically if a given functionality is provided by another smart contract the labeling could be used as a guidance for contract purpose.

The paper is structured as follows: In Section 2 we provide an overview of publications related to our work. A brief summary of the main clustering techniques used in our work is given in Section 3. The methodology used in our work, including how the dataset was built, is explained in Section 4 and a detailed report of our experimental results is provided in Section 5 followed by our conclusions and future work in Section 6.

2. Related Work

In the past few years cryptocurrencies such as Bitcoin, Peercoin, Litecoin and many others have started to be accepted as payment methods in more and more online services [2]. As an evolution of these cryptocurrencies, other blockchain based currencies started to emerge allowing the creation and deployment of smart contracts, like Ethereum [1] and Hawk [3].

Most of the research into cryptocurrencies such as Bitcoin and Ethereum has focused on either protocol improvements or security. In [4] the authors propose an upgrade to the Bitcoin blockchain to improve security and allow a larger transaction throughput. This improvement called GHOST for ”Greedy Heaviest-Observed Sub-Tree” was, in fact, implemented in multiple cryptocurrencies developed after Bitcoin including Ethereum [1].

The problem of malware detection is similar in some aspects to the problem of labeling contracts. In both cases one often has only compiled bytecode to work with. As such methods for malware detection can be utilized for smart contract identification. The work carried out in [5] and [6] make use of measuring the similarity through use of combinations of different distance measures/hashes to accurately label files as either malware or not. Both papers make use of the compiled files, or bytecode.

No prior work has addressed the large scale data analysis of smart contracts. The major source of information is currently `etherscan.io` , where all contracts can be explored and simple bytecode level metrics can identify similarities between compiled pieces of code. However, even if largely used to get some insights into smart contracts, the usage remains mostly driven by analyzing an individual contract and not a large scale global analytics in the overall contract space, which is the objective of our paper.

3. Background

In this section we provide a quick overview of the different clustering techniques used in our work and why they were selected. We had to leverage existing techniques, which do not assume prior knowledge of the number of clusters, or techniques which can efficiently deal with large number of data items in the presence of noise. This is due to the lack of any ground truth, as our work is exploratory and there are no current methods for automated analysis of contract purpose.

3.1. Clustering: Affinity Propagation

Affinity Propagation [7] is a clustering algorithm that identifies the most representative items within the data known as the exemplars by using a message passing algorithm to identify the fitness of a data point to represent its neighbors.

This method was selected because, unlike most clustering methods, it selects the number of clusters itself based on the data provided. A useful feature when dealing with a situation in which no ground truth is available. As with K-medoids, affinity propagation takes the most central data point in the cluster as the exemplar of that cluster.

3.2. Clustering: K-medoids

The second approach used was K-medoids. It is a partitioning based clustering algorithm which behaves in a similar manner to K-means clustering algorithm [8], with the difference that it takes most central data point as a medoid/exemplar rather than the mean of distances in the cluster. Using this point, the partitioning operates by minimizing the sum of differences between each of the data points and the reference point.

While affinity propagation does not require the number of clusters to be picked, K-medoids does. Cluster diameter was used to pick the appropriate value of k as discussed in section 4.1, as a result seven clusters were used.

4. Methodology

In this section we provide a summary of the methodology used in our work, starting with how our dataset was built, followed by the labeling of our contracts, then the hashing and distance measurements used. Lastly, some observations about the frequency distribution of name words in the dataset.

4.1. Dataset

The dataset used was created by taking all verified Ethereum contracts at the time of experimentation from `etherscan.io`. These contracts were selected because of the guaranteed availability of source code to aid in manual analysis, and the assumption that creators who have taken the time to upload the source code and verify their contracts online have a high likelihood of naming their contracts properly. That is to say, with a name which correspond to their function. This is important for this work as contract names form a part of our attempt to label clusters.

The dataset's size is slightly smaller than the original list of verified contracts. This is due to the use of `etherchain.org`'s API to acquire the bytecode for each contract. 62 out of 998 contracts had no bytecode returned by the API and were removed from the dataset as a result. At the time of writing the API is still in beta, we presume some flaw in it is responsible for lack of bytecode for certain contracts.

Our first step was creating clusters using the mean of three distance measurements to ascertain the similarity of *ssdeep* hashes of each contract's bytecode. The same measurements were used for both clustering method. K-medoids required the additional step of measuring the maximum diameter of all clusters for each value of k. From doing so we ascertained through inspection that $k = 7$ provided a good balance between number of clusters and a lowered maximum diameter.

Secondly, a group of *name words* was generated for each cluster, this process is explained in more detail below. The name words were then used to help identify the purpose of the contracts in a cluster.

4.2. Labeling

The labeling words were acquired by an automated process of taking contract names from the source code section of the page (it was discovered the 'Contract Name' tag above is sometimes left blank) for each contract on `etherscan.io`.

The names were then tokenized, which included accounting for various naming conventions such as CamelCase and snake_case. The resulting name words for each cluster were recorded. The four most frequent words for each cluster were used as label words to describe it. Four words were chosen in order to provide a good description of a cluster's purpose whilst reducing undesirable naming overlap between clusters.

A frequency distribution score was given to each cluster. The score was calculated as:

$$f(\alpha, \beta) = 1 - \frac{\alpha}{\beta}$$

Where α is the total number of unique name words and β is total number of name words. This gives a score between 0 and 1 where closer to 1 is indicative of a more homogeneous cluster.

4.3. Hashing Step & Distance Measurement

Contracts were represented by taking a hash of their bytecode using the non-cryptographic *ssdeep hash*, also known as context triggered piecewise hash (CTPH) [9]. This hashing function produces hashes that are uniform but, unlike cryptographic hashes, not random. This enables us to compare the similarity of the hashes to ascertain the similarity of the bytecode for contracts.

Similarity has been measured using the mean of three well known distance measures, levenshtein, jaccard and sorenson. The mean was taken in order to mitigate a potential inaccuracy in a measure by it being balanced out by the others, a method used effectively in [5] and [6].

4.4. Dataset Frequency Distribution

The frequency distribution of name words for the whole dataset shows some interesting trends in terms of contract purpose overall. Our methods produced a total of 1803 name words after formatting and tokenization was completed. The 10 most frequently used words are shown in the table 1.

TABLE 1: Frequency Distribution of the 10 most common Name Words

Word	Freq
token	102
dao	69
withdraw	56
dice	46
presale	34
factory	31
ether	25
ponzi	23
eth	21
asset	17

The words 'dao' and 'withdraw' pertain to DAO or the breach thereof, showing that a large proportion of `etherscan.io` verified contracts are still dominated by DAO related contracts. For the unaware reader, The Dao [10] was a decentralized organization and an implementation of an investor-directed venture capital fund. Because of a vulnerability exploited in 2016, attackers were able to steal one third of its funds which lead to a hard fork in the Ethereum blockchain. Even though the fork allowed for the recovery of funds, it is questionable to what extent this is not undermining the original purpose of having a blockchain for managing the trust.

The present of 'dice' and 'ponzi' suggests that there are a lot of verified gambling contracts active on the blockchain. The 'presale' word is fairly self-explanatory and shows a lot of the verified contracts have been or are being used for presales, what these contracts are presales for is beyond the scope of this paper to identify. Finally, the most used word, 'token', is ambiguous to some extent. Currency in Ethereum is often referred to as tokens, however because Ethereum allows for, and encourages, the use of custom tokens [11] it is not clear what the word token pertains to.

The high frequency of an amibguous word such as 'token' highlights that it may not always be possible to extract a semantic from name words and, therefore, contract purpose. This problem is compounded by the lack of ground truth from which to attribute purpose to clusters and contracts. Some potential improvements are discussed in section 6.

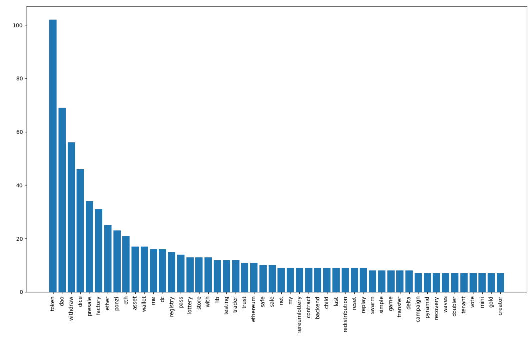


Figure 2: Frequency of 50 most common words

Fig 2 shows that distribution frequency is dominated by the first four words, with 'token' having a frequency double that of some of the closest words to it. It may be the case that the high presence of 'token' is obscuring the purpose of contracts as it shines no light on what a contract might be used for. Conversely, the words following 'token': 'dao' and 'withdraw' are also very common and are useful for identifying a contract's purpose.

It is interesting to note that the graph shows how the identification of gambling contracts is spread across a number of words, most prominently 'dice', 'ponzi' and 'lottery' with more words that can be associated with gambling appearing further down in the top 50.

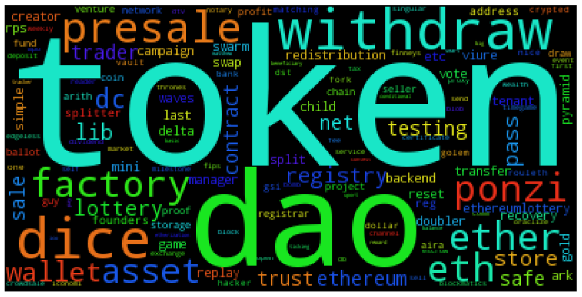


Figure 3: Word Cloud for all name words in dataset

5. Experimental Results

The analysis of results below involves identifying the purpose of a cluster through the use of its associated name words and the presence of a high transaction (TX) volume contract where transaction volume is taken to be *internal transactions* + *external transactions*. We take from the dataset the five contracts with the highest TX volume for the analysis. If a high TX volume contract is present in a cluster we look to see if the cluster labels can account for it. All of the top five high TX volume contracts are related to the DAO attack, summer 2016 [11].

For K-medoids the medoids (also called exemplars or centoids) for each cluster are looked at in terms of their function and discussed in comparison to the name words for the cluster and any high TX contracts that are present.

5.1. Affinity Propogation

The highest TX volume contract is found in a cluster with the name words 'split', 'safe', 'replay' and 'username'. With the exception of 'username', which only appears once in all 16 name words, it is apparent that this cluster contains 'ReplaySafeSplit' contracts, giving it a clear purpose. The 'ReplaySafeSplit' contracts exist to attempt to mitigate a type of replay attack.

The second 'ReplaySafeSplit' contract appears in a cluster whose three most frequent name words, in order of frequency, are: 'safe', 'replay' and 'split'. Given the presence of the high TX contract and the name words being almost exclusively the name words for the high TX contract this seemingly gives this cluster a clear purpose. Curiously, however, the third highest TX contract 'TheDAO' is also found here. It was not expected that these contracts would be found in the same cluster. The fourth highest TX contract is also found in this cluster which is in line with the second highest TX contract and the name words as it is also a 'ReplaySafeSplit' contract.

The fifth and final highest TX contract 'ManagedAccount' is found in a cluster with the labels 'dice', 'refund', 'ethereumlottery' and 'account'. The frequency distribution score for this cluster is 0.13 meaning there are almost no repeated words, the presence of the words ’lottery’ and ’dice’ suggest that it contains gambling contracts. This goes some way towards explaining the low score as there are a number of disparate words that can pertain to gambling contracts. These words are currently identified manually. However the present of the high TX contract indicates that the cluster also contains contracts with a separate DAO related purpose.

While affinity propagation has too many clusters to manual inspect them all, it is worth noting that in Fig. 4 that the two clusters with the greatest number of contracts also have the highest frequency distribution score. Both of these clusters’ name words consist almost exclusively of ’dao’ and ’withdraw’. This suggests that affinity propagation is able to identify DAO withdrawal contracts with a high level of accuracy. It also adds weight to the theory that affinity propagation has a number of separate clusters with the same purpose.

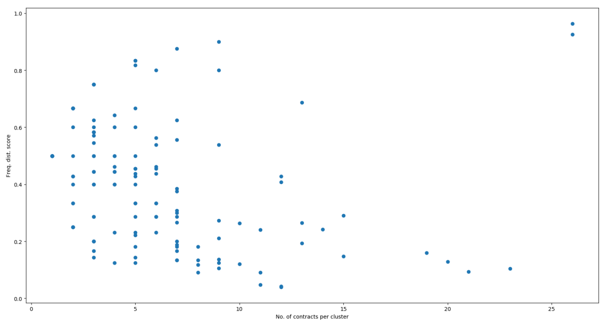


Figure 4: Cluster size x Freq. Dist. score

5.2. K-medoids

Out of the seven clusters four had an overall word frequency distribution score of >0.4 as can be seen in Fig. 5. We carried out manual verification on these clusters by taking the five contracts with the highest transaction (TX) volume and attempting to ascertain whether the clustering method was able to label them correctly.

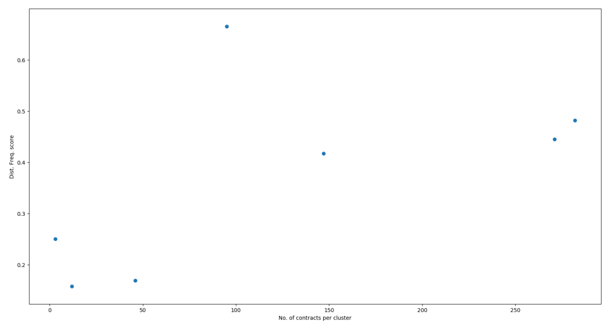


Figure 5: Cluster size x Freq. Dist. score

The first two highest TX contracts, both named ’ReplaySafeSplit’, were placed in the cluster which has the labels ’presale’, ’token’, ’dice’ and ’factory’. As such it is not immediately obvious that the labels apply to the ’ReplaySafeSplit’ contract. We conjecture that the size of this cluster indicates it is grouping together different kinds of contracts.

The third contract is called TheDAO, it was placed in the cluster with the labels ’dao’, ’withdraw’, ’token’ and ’mini’. The contract has been correctly labeled by the cluster. It is worth noting that the label ’withdraw’ also holds a strong likelihood of pertaining to TheDAO as well, many ’withdraw’ contracts were setup when TheDAO’s security breach took place.

The fourth is again named ’ReplaySafeSplit’, its cluster labels are ’dao’, ’withdraw’, ’token’ and ’mini’. While DAO withdrawal contracts are related to the safe split contracts in someways these name words do not obviously pertain to the ’ReplaySafeSplit’ contract.

The fifth and final contract is named ’ManagedAccount’, it forms part of The DAO. Its contract labels are ’token’, ’dice’, ’factory’ and ’presale’. Neither of the contract’s name words appear in the labels the words. As can be seen in section 4.4 ’token’ and ’factory’ are both in the top 10 common words, with ’token’ being the most common. ’factory’ most likely pertains to contracts that create other contracts and therefore tells us something about the purpose of clusters that have this label. On the other hand given that ’token’ refers to any currency on the blockchain, including the native ether, it tells us very little about the purpose of contracts in a cluster.

5.3. K-medoids Centoids

We looked at the medoid of each cluster, in an attempt to define a purpose for contracts in the cluster as well comparing it to the name words as a definition of cluster purpose.

The first cluster’s medoid is a contract called ’Presale’. This aligns well with the name words, the most common word for this cluster is ’presale’ closely followed by ’token’, although ’token’ is fairly ambiguous it may be the case that there these two name words are associated as presales often issue some kind of token. As such, it seems that the medoid and the name words correlate to label this cluster as ’presales’ contracts.

This second cluster’s medoid is called ’Contest’, the source code indicates it is a contract designed for voting. The most common name word for this cluster is ’contest’ which implies contracts with a similar function are to be found within it. It is worth noting however that this contract has a low frequency distribution score of 0.16, meaning that the contracts in this cluster are largely named differently.

The third cluster medoid, named ’WithdrawDAO’ is very clearly, as the name suggests, a contract for withdrawal of funds from DAO. The name words for this cluster strongly suggest that it contains contracts dealing with DAO withdrawals. The top two name words are ’dao’ and ’withdraw’, they comprise over half of all the name words for the cluster.

The fourth cluster medoid is named ’fairandeasy’ which is a gambling contract. This can be seen in the name words for this cluster to some extent. Although the frequency distribution score for this cluster is only 0.17 three of the four most common words are ’dice’, ’pyramid’ and ’simple’ which are all commonly used words for gambling contracts.

The fifth cluster’s medoid is named ’ProtectTheCastle’ and appears to be a gambling contract. One of the four most commons name words is ’dice’ which suggests that contracts in this cluster are also primarily gambling. However the most common word by a large majority is ’token’ which is not a name word that can be as clearly associated with gambling due to is ambiguity. The fact that one of the top five highest TX volume contracts, ’ManagedAccount’, is also found in this cluster suggests that it contains contracts that have a number of different purposes. As ’ManagedAccount’ is part of The DAO it suggests the cluster contains both gambling contracts and contracts that deal with DAO.

The sixth cluster is something of a special case, the medoid is called ’Eater’ and is a completely empty contract. The cluster contains only one other contract which also has no discernible purpose, containing only three variables and no operations. As such, it appears this cluster catches either empty or close to empty contracts with no purpose. No analysis of name words is required due to the low number of contracts.

The final cluster’s medoid is called ’Doubler’. Analysis of the source code suggests that this is a gambling contract. One of the four most common name words is ’dice’ which suggests that this cluster, as well as the fourth and fifth clusters, consists, at least in part, of gambling contracts. Moreover, this cluster follows the pattern seen in the fifth cluster of having ’token’ as its most common name word. Again, the presence of ’token’ does not tell us much about the purpose of the contracts.

6. Conclusion

From the results it can be seen that using clustering based on bytecode hash similarity it is possible to identify the purpose of a contract by applying labels to clusters and by using manual analysis of name words and high TX transactions to ascertain purpose. Although Affinity Propagation resulted in a high number of clusters (142), it was more accurate in grouping contracts with a similar purpose. It produced a number of separate clusters which had the same or similar name words and similarly high frequency distribution scores. This indicates that various clusters are identifying contracts with a similar purpose. The labels for these clusters therefore also end up being the same. In the future this could be used to help guide supervised learning methods by grouping contracts from similarity labeled clusters into one category. K-medoids has clustered accurately for its exemplars but obfuscates the purpose of high TX contracts. An interesting line for future work would be to look at the relationship between bytecode similarity and contracts grouped by transaction volume between each other. We have offered an insight into the purpose of verified contracts in the Ethereum system and paved the way towards a GUI-driven framework for contract labeling in the future.

In light of the high frequency of the ’token’ name word it would help to alleviate the problem of finding semantic meaning to selectively omit name words that are ambiguous in terms of the contract purpose. Doing this may well enable us to label more accurately. Fig. 2 highlights the usefulness of grouping together words with a similar meaning (in this case, gambling) for the purpose of being better able to label clusters. This method could also be used to modify the frequency distribution score. Whereby all words with the same meaning in terms of contract purpose could be translated to one word, ’gambling’ for example. While the above does not remove the ground truth problem it would help better define semantic meaning from name words. We plan to extend our work by looking into other smart contract platforms and validating our results for the Hyperledger platform [12] in order to compare our approach on two conceptually different platforms.

References

[1] V. Buterin, “Ethereum: A next-generation smart contract and decentralized application platform,” https://github.com/ethereum/wiki/wiki/White-Paper, 2013.

[2] D. Tapscott and A. Tapscott, *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Penguin, 2016.

[3] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts,” in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 839–858.

[4] Y. Sompolinsky and A. Zohar, “Accelerating bitcoin’s transaction processing. fast money grows on trees, not chains,” Cryptology ePrint Archive, Report 2013/881, 2013, http://eprint.iacr.org/.

[5] A. H. Sung, J. Xu, P. Chavez, and S. Mukkamala, “Static analyzer of vicious executables (save),” in *Computer Security Applications Conference, 2004. 20th Annual*. IEEE, 2004, pp. 326–334.

[6] A. P. Namanya, Q. K. A. Mirza, H. Al-Mohannadi, I. U. Awan, and J. F. P. Disso, “Detection of malicious portable executables using evidence combinational theory with fuzzy hashing,” in *Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference on*. IEEE, 2016, pp. 91–98.

[7] B. J. Frey and D. Dueck, “Clustering by passing messages between data points,” *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[8] K. Alsabti, S. Ranka, and V. Singh, “An efficient k-means clustering algorithm,” 1997.

[9] J. Kornblum, “Identifying almost identical files using context triggered piecewise hashing,” *Digital investigation*, vol. 3, pp. 91–97, 2006.

[10] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’16. New York, NY, USA: ACM, 2016, pp. 254–269. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978309

[11] V. Buterin. Critical update re: Dao vulnerability. Accessed: 7 April 2017. [Online]. Available: https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/

[12] C. Cachin, “Architecture of the hyperledger blockchain fabric,” https://www.zurich.ibm.com/dccl/paper/cachin_dccl.pdf, 2016.