Free Chain: Enabling Freedom of Expression through Public Blockchains

1st Israa Alsarsour
Department of Computer Science and Engineering
Qatar University
Doha, Qatar
ia1205702@qu.edu.qa

2nd Qutaibah Malluhi
Department of Computer Science and Engineering
Qatar University
Doha, Qatar
qmalluhi@qu.edu.qa

3rd Yongge Wang
Department of SIS
UNC Charlotte
USA
Yongge.wang@gmail.com

***Abstract*—Everyone should have the right to expression and opinion without interference. Nevertheless, Internet censorship is often misused to block freedom of speech. The distributed ledger technology provides a globally shared database that is geographically distributed and that cannot be controlled by a central authority. Blockchain is an emerging technology that enabled distributed ledgers and has recently been employed for building various types of applications. This paper demonstrates a unique application of blockchain technologies to create a social media platform that supports the freedom of expression. The paper adapts permissionless and public blockchains in order to leverage their advantages of providing an immutable and tamper-proof digital ledger. This study shows the blockchain potential for providing censorship-resistant social media platform. The paper presents and evaluates possible methods for building such system using the Bitcoin and Ethereum blockchain networks. After exploring both blockchains the result shows that Ethereum blockchain is much beneficial to use as a platform to allow freedom of speech.**

***Index Terms*—Blockchain, Ethereum, Bitcoin, Freedom of speech, Cybersecurity, Cryptocurrency**

I. Introduction

Social media allows communication between end users, which offers them the potential to express their opinions and interact with each other. This recognition of social media continues to grow at an exponential rate [1]. Recently, however, there has been an increased attack on the freedom of expression on social media, such as YouTube, Facebook, and Twitter. The censorship software monitors peoples online activities and prevents them from accessing unwanted content or delete what has been posted. According to Freedom House, a total of 68 countries suffered a net decline in political rights and civil liberties during 2018, with only 50 registering gains [9].

Furthermore, the existing methods to overcome censorship, such as domain fronting, have failed. Domain fronting allows hiding the real Internet communication and route it through a third-party website. However, governments succeeded in stopping this mechanism by blocking all the traffic to the third-party website. One example is the Israel government's clear censorship on Facebook posts, Wikipedia content and any activity/program that uncovers what is happening in Palestine. A program called A Jewish-American on Israels Fascism: No Hope for Change from Within was blocked in 28 nations [2] in April last year. The TV show was a study that shows the aggressive attitude Israelis frequently practice towards Palestinians. Another example of censorship is when Chinese government deleted a letter which was posted in popular platforms Weibo and WeChat [3]. The Letter requested an official investigation for a case of sexual assault of a female student by her university professor which led to her committing suicide. An anonymous user then published the letter to the Ethereum blockchain, which means it is now permanently stored in the public domain.

The freedom of speech became internationally protected in the Universal Declaration of Human Rights (UDHR), which was announced by the United Nations General Assembly in 1948. After the end of second world war [4] UDHR declared that ”Everyone has the right to freedom of opinion and expression; this right includes freedom to hold opinions without interference and to seek, receive and impart information and ideas through any media and regardless of frontiers” [5].

The aim of this paper is to address the issue of suppression of freedom of speech. Blockchain tackles this problem by using a permanent decentralized ledger that no central entity has control over. This method prevents data from being tampered with, because there is no single dominant authority that can alter data [6]. The technical objectives of our proposed research includes the following:

- To identify alternative methods of adding arbitrary data on public non-permissioned blockchains.
- To evaluate and compare the performance of the alternative methods and adopt one to use to enable freedom of expression.
- To build an optimized publishing tool based on the adopted method.
- To build a viewing tool that enables user-friendly exploration of content published on the blockchain.

Blockchain technology is becoming popular for supporting private and secure networks. While the main application of public non-permissioned blockchains is to record currency transactions, recently the thought of using them as a tool for free speech has emerged. Therefore, we aim to use blockchain to enable free speech in an efficient and effective environment in terms of cost, time as well as tools to view people's opinions protected from censorship by governments and other pressure groups, regardless of the political views being expressed.

The remainder of this paper is structured as follows; a background on the fundamentals of Bitcoin and Ethereum blockchains and the relevant topics is given in section II. After that, the methodology is given in section III where we discuss the different methods to insert data into the blockchains. In section IV, the results and discussions are presented. Finally, the conclusion and future work is given in section V.

II. Background

A. Bitcoin

There was a change in the world after bitcoin, a cryptographic currency, was introduced in the paper [7], written by a pseudonym "Satoshi Nakamoto". Since 2009, bitcoin has been widely adopted and used and is now one of the major cryptographic currencies on the market. The cryptography behind bitcoin is relatively simple. The start of the coin base by Satoshi Nakamoto is the binary string w_0. To mine the first bitcoin BTC, a random number 0 is required to be found so that the first two bits of $w_1 = H(w_0, r_0)$ is 00 (that is, $w_1 < 2^{|w_0|-2}$). Anyone finding this r0 is rewarded with a few BTCs. The next person to find another, so that the first two bits of $w_2 = H(w_1, r_1)$ is 00 is also rewarded with a few BTCs. This process continues and new blocks w_{i+1} 1 keep being added to the existing block chain $w_0, ..., w_i$. If the frequency of finding a BTC block is fewer than 10 minutes, the community starts a voting process to extend the required prefix of 0s in the hash outputs.

The bitcoin is a chain $w_0, w_1, ..., w_n$ where w_n is the current bitcoin head which everyone works on. The bitcoin network is a peer to peer (P2P) network and every participant works on the longest chain. There is no benefit to working on a shorter chain; the transaction included in these chains will be invalid. Bitcoin transactions are included in the hash inputs so that they can be verified at a later times. In particular, we have

$$w_i + 1 = H(w_i, TR, r_i)$$

where TR is the Merkle hash output of the transactions to be included and ri is a random number found to make r_i is a random number that one finds to make $w_i + 1$ have a specific number of 0s in its prefix. The Merkle hash tree is illustrated in Fig."1".

1) Transactions: In the bitcoin system, a user is identified by a public key (pubkey)of 26-35 alphanumeric characters presenting a bitcoin address [8]. A random 256 bit number is a private key, and an elliptic curve digital signature algorithm (ECDSA) produces the corresponding pubkey. A transaction is in the format of "Alice pays x BTC to Bob", presented by inputs and outputs illustrated in Fig. "2" . In our example an input is where Alice should provide a proof of the ownership to her x BTC received in a previous transaction by unlocking the x BTC calling a script scriptSig. An output contain the locking script called scriptPubKey holds the x BTC and Bob

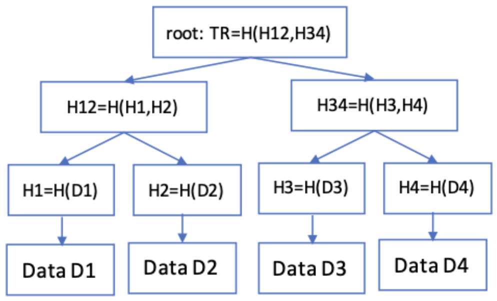


Fig. 1. Merkle hash tree.

pubkey address and only the receiver "Bob" will be able to unlock it using his signature and pubkey. Thus, a transaction is achieved by Alice signing the message "reference number, Bob's public key, BTC amount x", where the reference number refers to a block w_i in the current BTC chain w_i in the current BTC chain $w_0, w_1, ..., w_n$, where Alice received a minimum of x BTCs in a transaction with the provided reference number included in w_i. For instance, the block w_i ncludes a transaction with this reference numbers which shows that Alice received a particular quantity of BTCs.

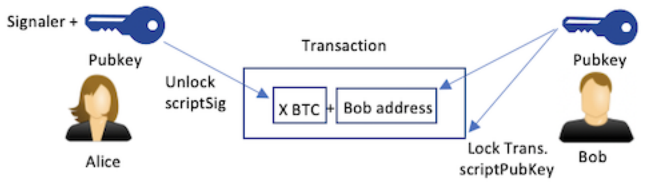


Fig. 2. Transaction in Bitcoin

2) Scripting Language: Transaction validation in Bitcoin is not an automatic process, it is achieved by executing Locking and Unlocking scripts which are placed in output and input scripts, respectively. In order to validate that a transaction redeems the output of another transaction correctly, both scripts should be combined and the result of the execution of both scripts should be true, else, the transaction would be invalid. Bitcoin scripting language is known as "Script", it is a simple Forth like programming language that was created especially for Bitcoin, what is special about Script is that it supports cryptographic operations for calculating the hash and verifying signatures. Script is stack-based, which means that the operations follow the operands. Another important characteristic of Script is that it is Turing-complete meaning that it is not a general-purpose programming language and it does not support loops and recursion, although this feature limits Bitcoin's abilities, it is considered as a security feature as the execution time and memory usage are predictable and it protects the transactions from logic bombs that might cause Denial of Service attack (DoS). Figure 3 shows how the locking and unlocking scripts are stacked [9]. A transaction is validated by executing the locking script of the previous

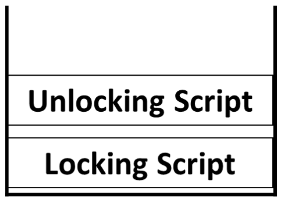


Fig. 3. Concatenation of Unlocking and Locking Scripts

transaction and unlocking script of the current transaction. Locking and Unlocking scripts are also known as scriptSig and scriptPubKey, respectively. The scriptSig is a spending condition of the output and the scriptPubkey is used to satisfy that condition .

3) Standard Transactions: A transaction is standard if it passes Bitcoin Core's IsStandard() and IsStandardTx() tests. Bitcoin support five standard transactions:

1) PAY-TO-PUBLIC-KEY-HASH (P2PKH): P2PKH is the default transaction in bitcoin Blockchain. The locking script known as ScriptPubKey is cosiest of hash of the public key of the receiver. To unlock (spent) the transaction a script of the receiver pubkey along with a valid digital signature should be provided known as ScriptSig.
2) PAY-TO-PUBLIC-KEY (P2PK): P2PK is a simpler version of P2PKH. The defference is that the locking script is now the pubkey instead of pubkey hash and the unlocking script is the only the signature.
3) PAY-TO-MULTISIG (Multisig): Multisig is a script where it has many **n** pubkeys as locking script and requires some or all **m** signatures to those pubkeys to unlock the script.
4) PAY-TO-SCRIPT-HASH (P2SH): P2SH is more advanced script used mostly for Multisig transactions. The locking script contains the hash of the custom redeem script. And to unlock it, the redeem script and the data needed to unlock it should be provided. It allow users to create their own redeem scripts, but also share them easily with others making the burden on the sender of the message to provide the redeem script and data needed to unlock (spend) it.
5) COINBASE: An input script used only by the miners is referred to as the "Coinbase". This data provides up to 100 bytes of arbitrary. For holding ASCII encoded string, for instance, names of their mining pools, miners are at liberty to manipulate these bytes of the Coinbase data. For example, the following text was added by Satoshi Nakamoto using the Coinbase data: "The Times 03/Jan/2009 Chancellor on the brink of the second bailout for banks" [10], when the Bitcoin blockchain was firstly used by Nakamoto. Whereas, this field is an approach to store arbitrary data in the Bitcoin Blockchain ledger, only miners have access to it and general Bitcoin users have no access to it. Thus, this section contains it for attention only, and, the same will not be mentioned for a second time.
6) OP_RETURN: The opcode OP_RETURN was introduced to Bitcoin blockchain as a reaction to the growing users who are trying to store arbitrary data in the blockchain by abusing other standard scripting opcodes such as P2PKH. opcode OP_RETURN can store 80 bytes of arbitrary data in each transaction. A transaction can have many outputs but only one OP_RETURN output transaction is allowed for the transaction to be considered standard [11]. OP_RETURN script always assess to false [12], thus creating unspendable UTXO. Miners can safely remove the output transaction of OP_RETURN from the UTXO set and do not need to keep track of them.

B. Ethereum

Although forth-like scripts in bitcoin are enough for designing different types of smart contracts, it has limited capability. An underlying philosophy in Ethereum is the inclusion of a Turing-complete programming language in the blockchain system is that various types of smart contract can be supported in the blockchain. The design of Ethereum was that of an Internet Service Platform with the objective of allowing anyone to upload programs to the Ethereum World Computer, and anyone can request that an uploaded program can be executed. Compared with Bitcoin, there two main functions of Ethereum:

- Ethereum is a blockchain with a built-in Turing-complete programming language, which allows anyone to write smart contracts and decentralised applications so that they can create their own ownership rules, transaction formats, and state transition functions.
- Bitcoin only supports "proof of work". Whereas Ethereum supports both "proof of stake" and "proof of work", where "proof of stake" calculates a node's weight as proportional to its currency holdings, rather than its computational resources.

The runtime environment for Ethereum smart contracts is based on the Ethereum Virtual Machine (EVM). The EVM can run any operation which has been created using the Solidity, a Turing-complete Ethereum scripting language. An Ethereum account is a 20-byte string consisting of four fields: nonce, ether balance, contract code (optional), and storage (empty by default). There are two types of Ethereum account: Ethereum Externally Owned Accounts (EOAs) and contract accounts. An EOA is linked to a private key and the only way to 'activate' a contract account is by using an EOA. A contract account is ruled by its internal smart contract code, which is programmed to be controlled by an EOA with a specific address. A smart contract program in a contract account is executed when a transaction is sent to that account. The transaction's sender has to pay for each step of the 'program' which they activate. This includes the costs of both computation and memory storage. The user is able to create a new contract by deploying code to the blockchain.

Ethereum also supports contracts that can generate logs through events, which are informative messages stored in each

Ethereum blocks transaction log, each event is associated to the address of the contract that triggered it.

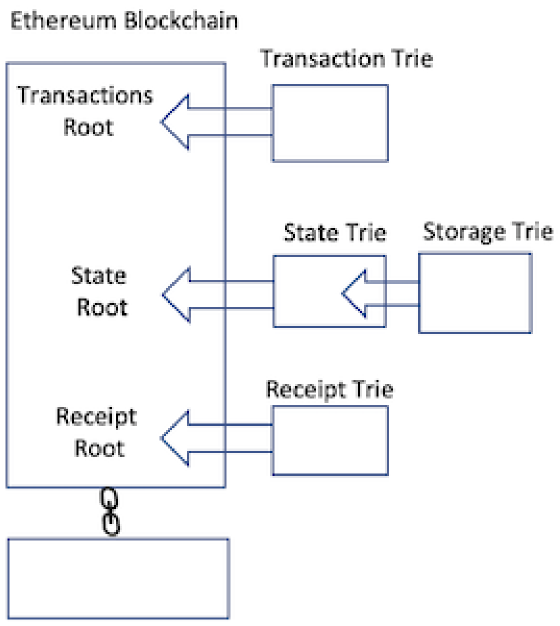


Fig. 4. Transaction Trie, State Trie and Receipts Trie in Ethereum

Ethereum has three Merkle Patricia trees, State, Transaction, and Receipt, in which each stores some type of data. The Receipt tree saves the information resulted from emitting an event log, such as block number, block hash, transaction hash, gas used by current transaction, logs which were created by the transaction and more. The Transaction tree contains parameters such as nonce, gas price, gas limit, recipient, transfer value, transaction signature values, and account initialization for contract creation and transaction data for message call [13]. The information of a transaction that deploys a contract get stored in a State tree, such as post-transaction state, the accumulative gas used, logs which were created from executing the transaction, and the information of these logs, which is stored in the bloom filter.

III. Methodology

In this section, we review different methods to insert data in Bitcoin and Ethereum blockchains.

A. Embed data in Bitcoin blocks

As we have discussed in Section 3, Bitcoin transactions are described using Forth-like scripts [14]. A script describes how the Bitcoin receiver can spend the received Bitcoins. For uncommon bitcoin transfers the script specifies that for Bob, the recipient, to spend the received bitcoins he has to present a public key and digital signature where the public key hash is equal to Bob's ID embedded in the script and the digital signature will prove the ownership of the private key which corresponds to the provided public key. A transaction is valid if failure is not triggered by the combined script, and top stack item is non-zero when the script exits. The stacks hold byte vectors of at most 520 bytes long. The script words are called opcodes (also known as commands or functions).

The goal of our proposed solution is to include a maximum number of non-transaction related bytes into the script without triggering failure so that the transaction will be included in the block by miners. There are several ways for arbitrary data such as short messages and files to be injected in Bitcoin ledger via several methods:

1) OP_RETURN: This opcode (op) is followed by one pushdata op which can be used to store 80 bytes of data that is not related to the transaction itself (e.g. human-readable messages.). This opcode considered non-standard to have more than one OP_RETURN output or an OP_RETURN output with more than one pushdata op in each transaction (see Fig. ”5” for a graphical illustration).

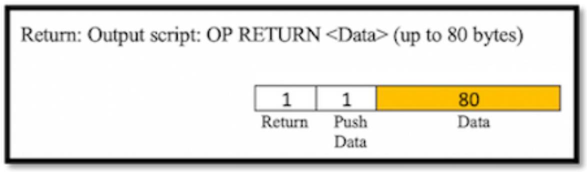


Fig. 5. Return Transaction

2) PAY-TO-PUBLIC-KEY-HASH: is a well-known and debatable data insertion technique, wherein, the data in the $< PubKeyHash >$ field of the output script is stored accompanied by a non-dust amount of Bitcoin. This is known as Pay-to- Fake-Key-Hash (P2FKH). There is no public key with the user, which can be hashed on the data being stored. Due to this, users can never spend these transaction outputs (see Fig. ”6” for a graphical illustration). Nevertheless, the miners are unaware if the hash denotes a real public key possessed by someone and as they are valid Unspent Transaction Outputs UTXOs, these UTXOs must be recorded (forever) by the miners. The P2FKH method provides the storage of 20 bytes per output; however, a single transaction can entail a number of outputs. In Bitcoin's blockchain, the images, text and mp3 files have been stored through this method and now it is this method utilized by tools such as Apertus.io [15].

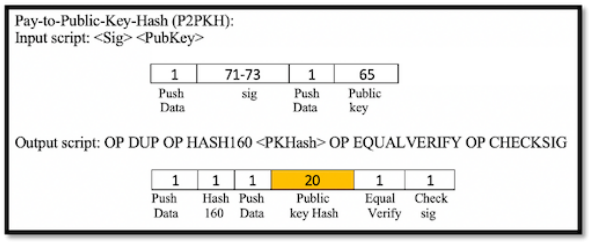

Pay-to-Public-Key-Hash (P2PKH):
Input script: <Sig> <PubKey>
1
71-73
1
65
Push Data
sig
Push Data
Public key
Output script: OP DUP OP HASH160 <PKHash> OP EQUALVERIFY OP CHECKSIG
1
1
1
20
1
1
Push Data
Hash 160
Push Data
Public key Hash
Equal Verify
Check sig

Fig. 6. Pay-to-Public-Key-Hash Transaction

3) PAY-TO-PUBLIC-KEY: Rather than a fake public key hash, one can store the data as a fake public key (P2FK). There are 65 bytes in an uncompressed public key, and three fewer OP codes make up the overall script, due to which, it is turned out as an efficient method for data storage as compared to the P2FKH (see Fig. ”7” for a graphical illustration). However, the community doesn’t use it as a widespread method to store the data. A likely reason for this is since the nodes can easily identify fake (uncompressed) public keys and this approach could be shut down by the miners in the future. In this regard, the data can be saved with a fake compressed public key (33 bytes) besides realizing data efficiency as compared to P2FKH.

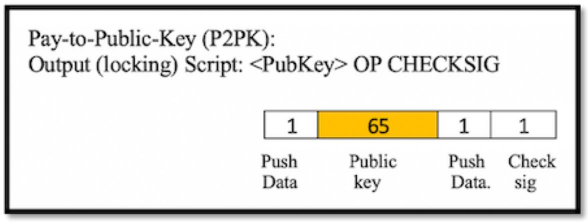

Pay-to-Public-Key (P2PK):
Output (locking) Script: <PubKey> OP CHECKSIG
1
65
1
1
Push Data
Public key
Push Data.
Check sig

Fig. 7. Pay-to-Public-Key Transaction

4) PAY-TO-MULTISIG: A one-of-two or one-of-three multisig script containing real public key and one or two fake keys having arbitrary data is another method to insert data (Pay-to-Fake-Multisig), which is generally seen in the Blockchain technology Creation of UTXO bloat can be prevented, since these transactions are spendable. As far as the lowest overhead cost is concerned, a (real) compressed public key would be used and the data will be stored via two fake uncompressed public keys (65 bytes each). With the help of this method, the UTXO set will maintain the data till the spending of these outputs (using the one real key) by the user. Within a single transaction, one can consistently store multiple P2FMS outputs in all of them using the same public key, as a result of which, data reconstruction becomes easy (see Fig. ”8” for a graphical illustration). In spite of that, there should be more than 400 bytes in the transactions that contain a single OP_CHECKMULTISIG. In-fact, 20 bytes per sigop is the default requirement, and 20 sigops are considered as one instance of OP_CHECKMULTISIG. The redemption of these UTXOs becomes unproductive because of these shortcomings. Additionally, as compared to the min non-dust values, the cost for spending these UTXOs will be greater. Thus, to store arbitrary data with a burn amount, users can use all three pubkey fields with no respect for the UTXO bloat.

5) PAY-TO-SCRIPT-HASH: Just like the P2FKH, data storage as a fake hash is done by Pay-to-Fake-Script-Hash (P2FSH) method. As compared to P2FKH, two fewer OP codes are required by the P2FSH (making

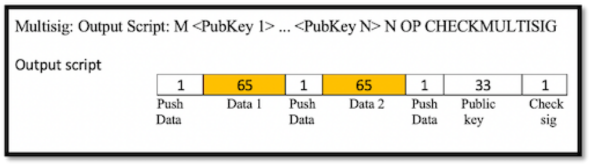


Fig. 8. Multisig Transaction

it more effective), however, an unspendable UTXO is still developed. The methods storing data in the input script, where a P2SH output is spent. Creating the UTXO and spending the UTXO are the two stages of the P2SH. A Redeem Script is first developed and the HASH160 algorithm is subsequently applied to this script for creation of the P2SH UTXO. accordingly, the output script is given below:

$$OP\_HASH160"RedeemScriptHash"OP\_EQUAL$$

For consuming this UTXO, an input script is created containing the Redeem Script (as a solo stack element, hence confined to 520 bytes), which is led by a series of Script operations, through which, the Redeem Script will conclude in only true upon execution. The data insertion has two approaches: arbitrary data may either be stored in the the Redeem Script, or/and may be stored in the bit of the input script leading the Redeem Script. E.g., a user can create a Redeem Script with an OP_PUSHDATA2 (three bytes) followed by a 517-bytes data element. Apart from OP_0, any stack element is considered as “true”, the UTXO will successfully be redeemed by this script. Despite this, owing to the 520-byte Redeem Script limit, large amounts of data can be efficiently stored in the segment of the input script leading the Redeem Script (see Fig. ”9” for a graphical illustration). Since June 2014, experts have been using the variations of the following P2SH-based methods for data storage in the Blockchain.

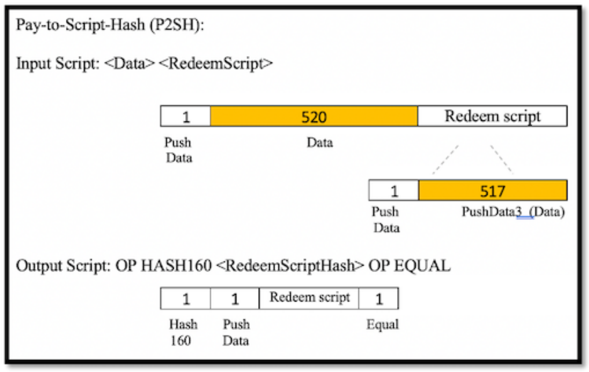

Pay-to-Script-Hash (P2SH):
Input Script: <Data> <RedeemScript>
1
520
Redeem script
Push Data
Data
1
517
Push Data
PushData3 (Data)
Output Script: OP HASH160 <RedeemScriptHash> OP EQUAL
1
1
Redeem script
1
Hash 160
Push Data
Equal

Fig. 9. Pay-to-Script-Hash Transaction

B. Embed data in Ethereum blocks

Ethereum has two types of transactions: message call transaction and contract creation transaction. Both types of transactions share the common fields: nonce, gasPrice, gasLimit, to, value, v, r, and s. Moreover, a contract creation transaction contains a field of an unlimited size byte array which specifies the EVM code for account initialisation procedure. Conversely, a message call transaction features a byte array data field of unlimited size which specifies the input data of the message call.

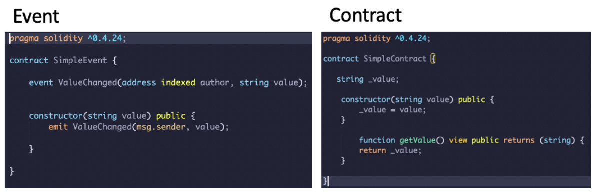
The objective of the indicated solution is to insert non-transaction related data into the transaction data field or unit, depending on whether the transaction is for a message call or contract creation. Therefore, we conduct an experiment using Ethereum Kovan Testnet to embed data to Ethereum Blockchain Using:

1) Data Field: A message call transaction can either be a call to a smart contract or be a simple transfer of ether to a non-contract "externally owned account". Whether the message call is a call to a smart contract is being called or it is a simple transfer of ether to a non-contract "externally owned account", the data files will be available for the senders to embed information of any size depending on their gasLimit. In the case of smart contract call the data filed is used to specify what function the parameters, which can be designed to hold the embedded data as the parameters. Similarly, if the call is a simple transfer of ether, then data field is unused and can hold the embedded data as well.
2) Contract Creation: The contract creation transaction field init contains the EVM-code. When creating a contract the EVM-code will be stored in the state Trie, which gives the user the chance to embed data in the EVM-code using parameterized constructor.
3) Event Creation: as we have discussed in section 3, events are messages stored in each Ethereum blocks transaction log. The EVM has logging functionality, this is actually what is behind those events. The event logs are part of the transaction receipts Trie (one of the three Merkle Patricia trees that are stored in Ethereum). This means Merkle proofs can be requested for log information because the receipt root is stored in the block. One can declare an event with a keyword event followed by event name and parameters.

$$eventMyEvent(uint_arg1, addressindexed_arg2)$$

Additionally, one can search for these logs later on in the chain if he/she want to go back in history and search for them. Events can inform external users that something happened on the blockchain and through returned values. Thus, applications can subscribe and listen to these events through the RPC interface of an Ethereum client.

Since the fields data and init are specified as unlimited size byte arrays, theoretically one can embed as much information as she wants in Ethereum transactions if she has sufficient ether (that is, more than gasLimit).



```
pragma solidity ^0.4.24;

contract SimpleEvent {

    event ValueChanged(address indexed author, string value);


    constructor(string value) public {
        emit ValueChanged(msg.sender, value);

    }

}
```

```
pragma solidity ^0.4.24;

contract SimpleContract {

  string _value;

   constructor(string value) public {
       _value = value;
   }

       function getValue() view public returns (string) {
       return _value;
   }

}
```

Fig. 10. Solidity Code of Event and Contract

IV. Results And Discussion

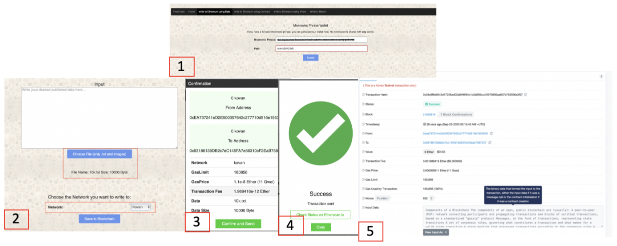
In this study, we built a system which allows the user to publish text data into the Ethereum blockchain. The user has the ability to choose between the different methods that we discussed in the preceding section. Additionally, the user has the ability to see the published data from different methods throughout the system as well as on the blockchain ledger. Finally, they key point of our system is that it is a standalone application which ensures that it would not need additional money to keep it floating.

We ran our experiments using macOS Catalina version 10.15.5 operating system. The processor of the machine is 2.8 GHz. We used Kovan test-net in Ethereum blockchain. Additionally, we used test-net network in Bitcoin network. We ran the experiments on different days during August 2020. Finally, Ether.js 4.0.0 API is used in Ethereum blockchain using Webstorm version 2019.3.4. We used bitcoin-0.11.3.jar in Bitcoin blockchain using Eclipse photon 2018 version 4.8.0.

To evaluate each method we used these parameters: The cost depends on the transaction fees times the number of transactions that are needed to publish data in the ledger. Since the rate of adding blocks is constant, the speed is determined by the number of transactions needed to insert data into the ledger and the number of blocks in which these transactions will be included. For example, if we have a message we need to publish, it requires **n** transactions with no control of how they will spread over blocks which might require **m** blocks. Because a block is added to the Bitcoin blockchain every 10 minutes [17] and every 15 seconds in Ethereum [18]. Finally, the ease of browsing was also evaluated. This is affected by the way we fragment the data and recollect it again to present it in the GUI interface that the users will be using to publish and browse their data.

A. Evaluating Ethereuem

Gas limit is about 10M gas, leading to typical block sizes of 20-30KB. Therefore, we tried 1KB, 10KB, 20KB, and 30KB and 40KB data sizes to experiment inserting data into the ledger. As we discussed in the previous sections, Ethereum has three methods to insert data in the ledger. For the data field, using Ether.js API, we were able to insert any size of data as long as we had sufficient gas. We inserted data by making message calls to a new address each time. The value field was set to zero, but we specified the gas price based on the appropriate gas price currently in the network. Even though we ran our experiments on Kovan, the users have the ability to choose which network they want to publish to. An estimated transaction fee pops up to the user before sending the transaction for them to confirm the transaction. After the confirmation, a link to where the transaction is in the ledger (transaction hash) is provided for the users to see their data in the ledger. They have the option to also explore the transaction in the system after providing the transaction hash see Fig "Fig. 11" for clear steps of using our system.



If you have a 12 word mnemonic phrase, you can generate your wallet here. No information is shared with any server.

File Name: 10k.txt Size: 10000 Byte

Choose the Network you want to write to:
Network: Kovan

0 kovan
From Address
0xEA737241eD2E500357642c277710d516e195

0 kovan
To Address
0x93186139DB2c7eC145FA7e56310cF3EaB758

Network	kovan
GasLimit	180856
GasPrice	1.1e-8 Ether (11 Gwei)
Transaction Fee	1.989416e-12 Ether
Data	10k.txt
Data Size	10000 Byte

[This is a Kovan Testnet transaction only]

Transaction Hash:	0x34c6f6e6043d77256ad02a849694c1c3d056cccf3979800aa607b763506a3f07
Status:	Success
Block:	21093918 7 Block Confirmations
Timestamp:	26 secs ago (Sep-23-2020 02:15:40 AM +UTC)
From:	0xea737241ed2e500357642c277710d516e1953648
To:	0x93186139db2c7ec145fa7e56310cf3eab7587237
Value:	0 Ether ($0.00)
Transaction Fee:	0.001989416 Ether ($0.000000)
Gas Price:	0.000000011 Ether (11 Gwei)
Gas Limit:	180,856
Gas Used by Transaction:	180,856 (100%)
Nonce Position	832 0
Input Data:	Components of a Blockchain The components of an open, public blockchain are (usually): A peer-to-peer (P2P) network connecting participants and propagating transactions and blocks of verified transactions, based on a standardized "gossip" protocol Messages, in the form of transactions, representing state transitions A set of consensus rules, governing what constitutes a transaction and what makes for a

The binary data that formed the input to the transaction, either the input data if it was a message call or the contract initialisation if it was a contract creation

Fig. 11. Example Transaction

For inserting data using contract creation method, we created a contract that has a string field to hold data and it is updated using a parameterized constructor using solidity. Thus, when we deployed the contract in our system we included the data in the constructor which allowed data to be included in the contract EVM-code as well as in the state Trie. The last method we used is contract creation using events. We created a contract using solidity that saved data in an event through emitting it using parameterized constructor which saved the data in the contract EVM-code as well as in the transaction receipt Trie.

We ran the experiments multiple times and each point in Figure 12 is a reading of 5 times. In each run we set the gas price differently which did not affect the gas used results. We calculated the transaction fee each run which varies because of the following equation.

$$TransactionFee = gasPrice * gasUsed$$

Figure 13 illustrates the transaction fee in Ether at 0.000000011 Ether (11 Gwei) gas price.

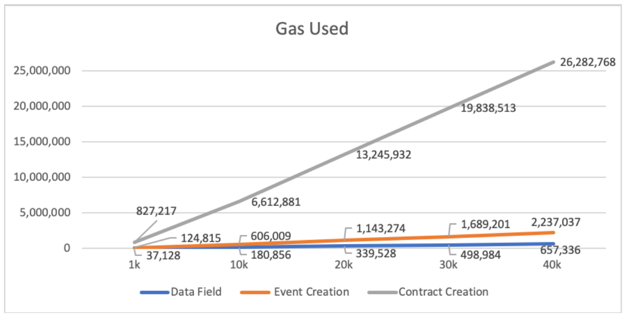


Fig. 12. Gas Used Ethereum

In Figure 12, we can see that data field is the least expensive method to be used to insert data into the ledger in terms of gas used which leads to cheap transaction fee as well. Then, contract creation-event methods cost and transaction fee are a little bit higher from field data method since the data is stored in the receipt trie as logs and in the init data field along with the EVM-code in the transaction tries. Lastly, contract creation method is the most expensive one as shown in Figure 13, since the data is stored in the state and transaction tries which is much expensive than storing data in other tries. Therefore, if a user wants to insert data to Ethereum blockchain it is recommended to use either of the data field methods.

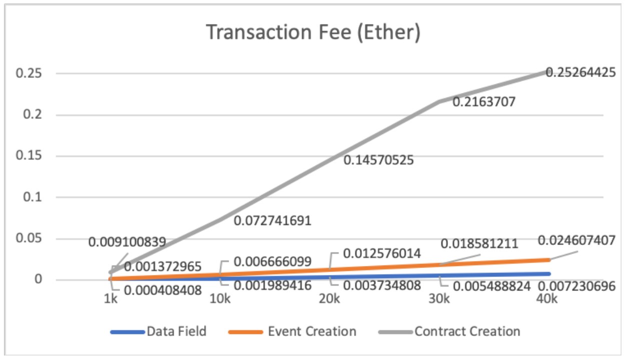


Fig. 13. Transaction Fee Ethereum

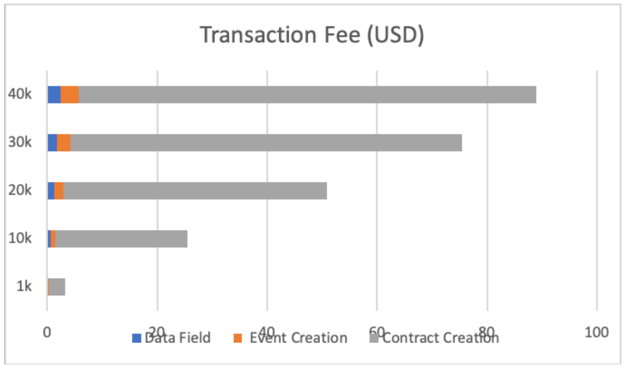


Fig. 14. Transaction Fee Ethereum (USD)

B. Evaluating Bitcoin

Bitcoin hard forks have splitted Bitcoin into the following cryptocurrencies: Bitcoin Core (BTC, the original version by Satoshi Nakamoto), Bitcoin Cash (BCH), Bitcoin Gold (BTG), and Bitcoin Private (BTCP). Currently, Bitcoin Core has a block size limit of 1MB and Bitcoin Cash has a block size limit of 32MB. In the following, we only consider the data bandwidth for Bitcoin Core.

In the Bitcoin Core implementation, there is no size limit on Bitcoin transactions. Thus the transaction size is essentially limited by the block size. However, there is a transaction fees [16] document mentioned that "Then transactions that pay a fee of at least 0.00001 BTC/kb are added to the block, highest-fee-per-kilobyte transactions first, until the block is not more than 750,000 bytes big." In the Bitcoin Core source code, the maximum allowed script size is 10,000 bytes. However, there is no limit on output per transaction. Theoretically one can put as much as 750KB data in one transaction, though one may use a more smart strategy to split the data into multiple transactions.

To make the comparison between both blockchains much easier, we considered using 1KB, 10KB, 20KB, and 30KB and 40KB data sizes to experiment inserting data into the Bitcoin ledger as well. We ran the experiments using bitcoin.js API. As we have mentioned in the preceding sections, bitcoin has five different methods to insert data in the ledger. One is allowed to use multiple methods within one script with the restriction that each method can not exceed the data allowed limit which was explained in the previous section through figures. For the chosen data sizes, when we want to use P2FK method, we have to divide the data into segments of 65 bytes and use multiple P2FK in one transaction using multiple outputs. This is done to all other transactions methods except OP_RETURN.

Figure 15 and 16 illustrate the transaction fees in USD of each method needed to insert 1KB, 10KB, 20KB, and 30KB and 40KB data sizes. As we can see, comparing the transactions fee, the most expensive is P2FKH followed by OP_RETURN and P2FK. PAY-TO-SCRIPT-HASH and PAY-To-MULTISIG which is considered the least expensive method. Thus, if the user wants to insert data to Bitcoin ledger they are recommended to use PAY-To-MULTISIG method and PAY-TO-SCRIPT-HASH.

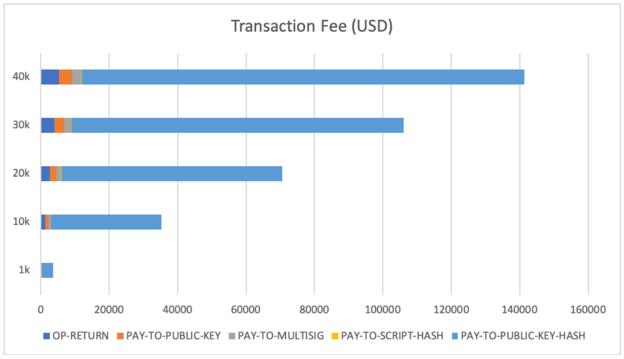


Fig. 15. Transaction Fee Bitcoin

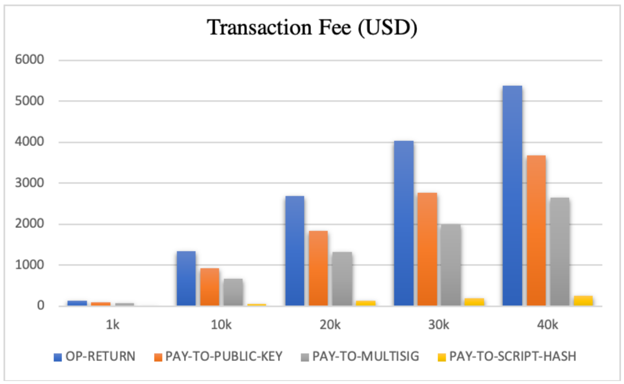


Fig. 16. Transaction Fee Bitcoin Without P2FKH

Another aspect to evaluate the methods are aiming at using fake keys to insert data by bloating the UTXO set. For example, P2FKH and P2FSH are considered fruitless by bloating the ledger with unspendable UTXO per 20 bytes of data. Another method considered fruitless is P2FK by providing only 65 bytes per one unspendable UTXO. One the other hand, P2FMS inserts 195 bytes using all three fake keys which result in unspendable UTXO. However, if one used two fake keys to insert 130 bytes and one real key that makes the UTXO spendable. Lastly, OP_PRTURN provides 80 bytes with unspendable UTXO but without bloating the set since the users can discard these UTXO from their sets.

C. *Comparison Between both Blockchain*

After looking at both results in Bitcoin and Ethereum blockchains, we can see clearly that Ethereum blockchain is much beneficial to use it as platform for allowing freedom of speech. Another aspect to look at it, as well, is reconstructing the data after inserting them in both ledgers. Since the number of transactions is much less in Ethereum blockchain than in Bitcoin, thus reconstructing the data is much easier.

V. Conclusion and Future work

Most countries with dictatorship governments deploy powerful firewalls to block contents from the external world and impose censorship on Internet media. For Proof of Work and Proof of Stake based blockchains, the content in the blocks are nonremovable. Furthermore, the participants of blockchains may keep their identity anonymous. Thus, blockchains may provide citizens within these countries with freedom of speech. On the other hand, criminals may use blockchains to distribute illegal contents (e.g., child pornography) or to establish black markets. It is noted that Bitcoins did not receive sufficient attention until it was used as the payment method on Silkroad for illegal contents. Though a blockchain itself has a limited bandwidth for content distribution, it is sufficient to carry out these illegal activities. When blockchains are used for these purposes, there is a chance that most countries will ban it. In this study, we have successfully proposed a solution that aims to prove and facilitate the ability to achieve freedom of speech by using Ethereum and Bitcoin blockchains. This is mainly based on blockchains immutable nature to prevent any changes in the data. This paper presented different methods to achieve that in both blockchains and presented the evaluation of the results of the proposed solution.

As discussed earlier, that blockchain has the potential to be used as a tool to achieve 'freedom of speech' due to the privacy and immutability features, it also has some challenges:

- Privacy: Recently, the state-of-art became active regarding De-anonymization of blockchain users. Moreover, there are some reported incidents of identifying some users in the blockchain. Thus, we need to use mixers to maximize the privacy of the users of our application although it will increase the cost.
- Risks of Arbitrary Blockchain Content: Since the proposed solution introduces the idea of inserting arbitrary

data into the blockchain, there is no guarantee that users will not misuse these channels for content insertion. One solution might be creating a blacklist of words that are agreed upon and shared in the blockchain to prevent the misbehaving people from publishing inappropriate content.

References

[1] G. Matthew, and A. Kohirkar, "Social media analytics: Techniques and insights for extracting business value out of social media," IBM Press, 2015.

[2] I. Bentov, A. Gabizon, and A. Mizrahi. Cryptocurrencies without proof of work. In International Conference on Financial Cryptography and Data Security, pages 142–157. Springer, 2016.

[3] E. Muzzy, How the Ethereum Blockchain Became a Tool in the Fight for China's #MeToo Movement. 2018, https://medium.com/@everett.muzzy/how-the-ethereum-blockchain-became-a-tool-in-the-fight-for-chinas-metoo-movement-e4017b1acddd

[4] Freedom in the World 2019 Democracy in Retreat. https://freedomhouse.org/report/freedom-world/2019/democracy-retrea

[5] Universal Declaration of Human Rights. http://www.un.org/en/universal-declaration-human-rights

[6] R. Hanifatunnisa, and B. Rahardjo. Blockchain based e-voting recording system design. 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA), 6.,2017

[7] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[8] A. M. Antonopoulos, Mastering Bitcoin: Unlocking Digital Crypto-Currencies. O'Reilly Media, Inc., 2014.

[9] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, Bitcoin and cryptocurrency technologies: a comprehensive introduction. Princeton and Oxford: Princeton University Press, 2016.

[10] J. Thomason, S. Bernhardt, T. Kansara and N. Cooper. "Blockchain Introduced." Blockchain Technology for Global Social Change. IGI Global, 2019. 25-59. Web. 4 Mar. 2020. doi:10.4018/978-1-5225-9578-6.ch002

[11] S. Bistarelli, I. Mercanti, and F. Santini. "An analysis of non-standard bitcoin transactions." 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). IEEE, 2018.

[12] M. Bartoletti, and L. Pompianu. "An analysis of Bitcoin OP_RETURN metadata." International Conference on Financial Cryptography and Data Security. Springer, Cham, 2017.

[13] G. Wood. "Ethereum: A secure decentralised generalised transaction ledger". Ethereum project yellow paper, 151(2014), 1-32. 2014

[14] Bitcoinwiki. Bitcoin script, https://en.bitcoin.it/wiki/Script, 2017.

[15] A. Sward, I. Vecna, and F. Stonedahl. "Data insertion in bitcoin's blockchain." Ledger 3 (2018). Harvard

[16] Bitcoinwiki. Bitcoin transaction fees, https://en.bitcoin.it/wiki/Transaction_fees, 2017.

[17] https://bitcoin.org/en/developer-guide#block-chain

[18] https://www.ethereum.org/ether