electronics

MDPI

Article

Practical Time-Release Blockchain

Sang-Wuk Chae, Jae-Ik Kim and Yongsu Park *

Department of Computer Science, Hanyang University, Wangshimriro 222, Seongdong-gu, Seoul 04763, Korea; sangoou@naver.com (S.-W.C.); wodlr9015@naver.com (J.-I.K.)

* Correspondence: yongsu@hanyang.ac.kr

Received: 1 April 2020; Accepted: 17 April 2020; Published: 20 April 2020

Abstract: Time-release cryptography is a special encryption technique that allows a message to be hidden for some time. The previous schemes have shortcomings in that the encryptor should predict the decryptor's computing power precisely or the trusted agent should be always available. In this paper, we propose a new, practical time-release blockchain, and find the key to decrypt the content after a certain time. In order to verify the effectiveness of the blockchain system automatically, which uses the proof-of-work (PoW) and the consensus algorithm in the the proposed technique, we have implemented a prototype version of our blockchain system using Python. The proposed method has the following advantages. First, the decryption time is automatically adjusted, even if the miner's computing power changes over time. Second, unlike previous time-lock puzzle schemes, our algorithm does not require additional computation work for solving the puzzle. Third, our scheme does not need any trusted agents (third parties). Fourth, the proposed method uses standard cryptographic algorithms.

Keywords: blockchain; cryptocurrency; e-voting protocol; network security; time-release cryptography

1. Introduction

Time-release cryptography can be classified into two categories [1]. The first is called "time-lock puzzles" whereby the encryptor assumes that the decryptor will continue to perform non-parallel operations to solve puzzles. The encryptor creates a computational puzzle that requires a precise amount of time to solve. The solution to the puzzle reveals a key that can be used to decrypt the encrypted information. The second is called "trusted decryption agents," where trusted agents provide keys to decryptors for decryption on specified dates [2]. However, the former method is disadvantageous in that it requires too much computation and the encryptor should predict the decryptor's computing power precisely. The latter method has the disadvantage that the trusted agent should be always available.

In this paper, we focus on time-lock puzzles. To overcome the shortcomings of the time-lock puzzles, we present a new and practical time-release blockchain, which uses the proof-of-work (PoW) and the consensus algorithm in the blockchain system to automatically find the key to decrypt the content after a certain time.

To estimate feasibility, we have implemented a prototype version of our blockchain system using Python. Also, we give an example for using our scheme: an e-voting system using the time-release blockchain.

The proposed method has the following advantages. First, by linking the difficulty of the blockchain and the difficulty of the time-lock puzzle, the difficulty is automatically adjusted even if the miner's computing power changes over time. Second, unlike previous time-lock puzzles, our scheme does not require additional computation work for solving the puzzle. Third, unlike the trusted decryption agent schemes, our scheme does not need any trusted third parties. Fourth, the

proposed method uses standard cryptographic algorithms whereas the previous work relies on complex non-standard algorithms.

This paper is organized as follows. Section 2 briefly explains the blockchain structure, the electronic voting system using the blockchain, and time-release cryptography. In Section 3, we propose our new scheme: time-release blockchain. In Section 4, we deal with prototype implementation of the time-release blockchain. In Section 5, we show a practical example for using time-release blockchain: e-voting system. In Section 6, we provide security analysis and conclude in Section 7.

2. Background

In Section 2.1, we briefly describe the blockchain system and in Section 2.2, we show major recent research efforts on the e-voting protocols using the blockchain. In Section 2.3, we summarize recent work on time-release cryptography.

2.1. Blockchain

The blockchain, invented by Satoshi Nakamoto in 2008 [3,4] can be considered as a distributed data storage that stores data in a growing list of records, which are called blocks. The blockchain connects them in linked lists using cryptographic primitives, and replicates and stores them over a large number of computers. Alternatively, the blockchain can be viewed as a (public) distributed ledger: instead of keeping transaction records on a centralized server, the transaction history is shared among all users in the network. By sharing and collating information with all participants using cryptographic primitives, the blockchain provides integrity of transactions such that transactions are not allowed to be forged or altered.

The blockchain can contain diverse information and can be utilized in a wide range of fields. Even though up to now it has been mainly used for a distributed ledger for cryptocurrencies, people are trying to use it in other business areas, e.g., electronic payments and digital certifications, tracking the entire process from cargo tracking systems, P2P lending, origin to distribution, authenticity of art, anti-counterfeiting, electronic voting, electronic citizenship, vehicle sharing, real estate registration, and medical records management.

First, we show the data structure of the blockchain and then we will briefly explain transactions and chained transactions structure. Even though we focus on the explanation of Bitcoin, which is the biggest blockchain network [3,5,6], other blockchain systems are roughly similar to Bitcoin.

2.1.1. Data Structure of the Block

The single block contains transactions information and is linked to the previous block using the cryptographic one-way hash function. A blockchain can be implemented in the form of a simple flat file or in a database [7].

A new block is created by referring to the previous block (i.e., the parent block). To do so, a "previous header hash" field in the block header refers to the previous block by containing the hash of the previous block. That is, each block and the previous block are connected using a cryptographic one-way hash function to form a chain structure.

While each block has only one parent block, it can temporarily hold multiple child blocks. This is a temporary phenomenon that occurs when new child blocks are produced concurrently by several nodes. Among them, only one block will become a part of the blockchain, and the temporary fork will disappear. The header and body structure of the block are in Tables 1 and 2, respectively. (In Section 3, we modify these and put additional fields to support time-release cryptography.)

Table 1. Structure of the block header.

Size	Field	Description
4 Bytes	Version	Version number for tracking software or for upgrading the protocol

4 Bytes	Timestamp	Approximate time when the block was created (in Unix timestamps).
32 Bytes	Merkle root	Root of the Merkle Tree [8] generated from the transactions in the block
32 Bytes	Previous header hash	A reference to the hash of the previous (parent) block
4 Bytes	Difficulty	The difficulty value in Proof-of-Work algorithm for this block
4 Bytes	Nonce	A counter value that makes the block header hash less than or equal to the target value

Table 2. Structure of the body in the block.

Field	Description
Transaction count	Number of transactions in the block body
Coinbase transaction	The first transaction in a block. It is always created by the miner and it has no inputs. This is the transaction that rewards the miner.
Transactions (tx_1, tx_2, … tx_n)	n transactions in the block

2.1.2. Chained Blocks Structure

The first block in the blockchain is called the genesis block. The genesis block is hardcoded in the client software, so every node starts with a blockchain consisting of at least one block. The genesis block is immutable and contains the proof of the date that the block was created.

The client has a local copy of the chain of blocks starting from the genesis block (Figure 1). If the node discovers a new block, the node adds it in the local copy and expands the chain. In this way, the local copy of the blockchain is constantly updated. After the node receives the newly created blocks from the network, the node checks the validity of the received blocks and connects the blocks to the existing chain of blocks.

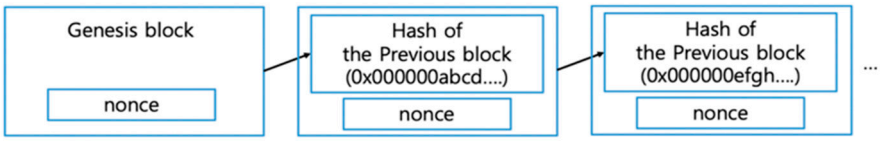

Figure 1. An example for the chained blocks structure.

2.1.3. Mining and Consensus

Mining is one of the major processes in the blockchain. Each node validates the transaction by the mining work. Also, it allows consensus to be made across the network without a central authority. Rewards from mining motivate miners' actions to keep the network secure. Decentralized consensus protocol [9,10] consists of the following processes.

1. Transaction verification: After receiving a new transaction, the node checks whether it is valid or not.
2. Create a new block: From valid transactions, the node tries to make a valid block. To do this, it conducts the mining work using the proof-of-work algorithm.
3. Managing the chain of blocks: The blockchain system is designed such that the longest chain of blocks is selected by the majority of nodes.

Verifying transactions: After receiving propagated transactions, the nodes verify them. Successfully verified transactions are stored in the transaction pool. The verification procedure includes the validity of syntax and semantics of transactions, which we omit in detail due to lack of space.

New block creation: First, the node makes the body of the block using valid transactions. Then, to construct the block header, the mining node must fill six fields in the header. Among them, nonce is related with the proof-of-work, i.e., the candidate block becomes a valid block by the proof-of-work algorithm: mining is to get a nonce value that makes the hash of the block header than the target value using Equation (1), where *SHA*256() [11] is the standard cryptographic one-way hash function.

$$SHA256(SHA256(\text{block\_header})) \leq 0^{l} || 1^{256-l} \quad (1)$$

In Bitcoin, difficulty is a measure of how difficult it is to find a hash below a given target. This value is used to control the generation time of the block constantly. In Bitcoin, blocks are generated on an average of 10 minutes and the difficulty is adjusted every 2016 blocks. Difold, Timeactual, and Timepredict are the current difficulty value, the elapsed time to mining 2016 blocks, and the expected time for mining 2016 blocks, respectively. Then, a new difficulty Dif_{new} is calculated using Equation (2), where we omit the detailed procedure for lack of space.

$$Dif_{new} = Dif_{old} \cdot (Time_{actual} / Time_{predict}) \quad (2)$$

Maintaining the chain of blocks: Each node maintains a chain of blocks, where each block containing a hash of the previous block up to the genesis block of the chain. The longest chain rule [4] means that individual nodes must accept the longest chain of blocks as the valid version of the blockchain.

Since there is no strict synchronization protocol that exists in the blockchain, each node has its own longest chain and sometimes there is a case whereby some minor nodes extend the chain that is different from the main chain (of majority nodes). They call that blockchain (temporarily) forked.

Figure 2 shows an example for the fork where the main chain for majority nodes is (B_{n-2}, B_{n-1}, B_n, B_{n+1}, B_{n+2}, B_{n+3}, B_{n+4}) and the forked chain (for some minor nodes) is (B_{n-2}, B_{n-1}, B_n, B'_{n+1}, B'_{n+2}). Due to the longest chain rule, as time goes on, the minor nodes for the forked chain will gradually have to accept the main chain, and eventually no node supports the forked chain. Generally, the procedure is more complex, but we omit the detail for lack of space. For further information, refer to [12].

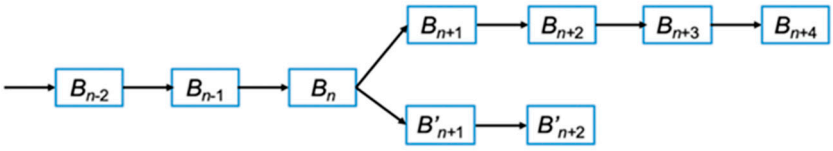

Figure 2. An example for the forked chain structure.

2.2. E-voting systems with the blockchain

Recently, the research on the e-voting protocol using the blockchain has been actively done until now [13–17]. By using the blockchain as a decentralized database, the transparency and integrity of the blockchain can be used to prevent forgery and to easily verify the vote counting. In the case of e-voting using a public blockchain where anyone can see the contents of the block, various techniques [18,19] are used to ensure the anonymity of voters and to prevent counting intermediate results during voting. The requirements for electronic voting using the blockchain are as follows [20]:

- Eligibility: Only eligible people can vote.
- Privacy: It must not be known how a specific person voted or to whom the specific voter voted.
- Coercion resistance: It must not be able to enforce voters to vote to the specific candidate. Also, there is no way to distinguish whether voters have voted as ordered or not.
- Physical verification: It must be checked whether someone stole another voter's identity or not.
- Forgiveness: Voters can change their vote before the voting procedure ends. It is related to coercion resistance; even if the voters are forced to vote, they can change the votes later.

- Verifiability: Every voter can verify the result that his/her vote is included correctly in the voting result. This brings transparency to the voting protocol.
- Immutability: After the voting procedure ends, voters' votes must not be changed.

In [13], the authors think that the voting protocol should not use a single key for security because anyone with the key can decrypt an encrypted ballot and access to the content of the ballot at any time. They proposed the protocol that uses Shamir's secret sharing [18] to distribute and store keys. Each candidate's identity is represented by a unique prime number, and voters cast their votes using ElGamal encryption. ElGamal encryption is homomorphic for multiplication. Hence, multiplying all the encrypted data and then decrypting and factoring it allows us to aggregate voting results without decrypting each vote.

BroncoVote [14], which is designed for university environments, is based on smart contracts of Ethereum. Also, it uses homomorphic encryption. The protocol briefly works as follows. First, voters vote using their own email where the votes are encrypted (using the server) and then sent to the voting contractor. The contractor in the blockchain homomorphically adds to the current number of votes. Homomorphic decryption is used to aggregate the results without decrypting all the votes.

Ranked choice voting [15] has multiple authorities, each with its own public and private keys. In this protocol, a new public key is derived from the public key of all authorities. This key is used to encrypt the votes. In the process of counting votes, all authorities cooperate to compute the private key and to decrypt the encrypted votes.

Follow My Vote [16] is the voting protocol where currently it is not fully integrated with the blockchain. It encrypts the data using symmetric cryptography. After voting and casting is done, the result is sent to the blockchain. Then, each voter is able to see his/her vote counted in the ballot box. Additionally, voters can watch the election progress in real time as casting votes are being processed. With full integration with the blockchain, it provides anonymity because it identifies the user by using his/her wallet address, and in the blockchain there is no link between the wallet address and user's identity.

BitCongress [16] uses proof-of-work and proof-of-tally to maintain data integrity. For authentication the user signs his/her vote with digital signatures and then encrypts the signed vote with candidate's public key. BitCongress uses a new key pair for an election to make it difficult to track data.

Some of the above voting protocols use homomorphic encryption to decrypt votes to reduce the time and cost of tallying. The common problem of the above protocols is that somebody should do the tallying work. This is a fatal obstacle to be fully decentralized. Reliance on the trusted third party raises concerns about privacy. With our time-release blockchain, many distributed systems (e.g., e-voting) can be decentralized, which is exemplified in Section 5.

2.3. Time-release Cryptography

The goal of time-release cryptography is to "send a message into the future." To do this, the sender must encrypt the message such that it cannot be decrypted until a certain time. Two representative ways to do this are "time-lock puzzles" and "trusted decryption agents[2]."

Time lock puzzles: This method assumes that the decryptor will continue to perform non-parallel operations to solve puzzle. The encryptor creates a computational puzzle that requires a precise amount of time to solve. The solution to the puzzle reveals the key that can be used to decrypt the encrypted information. This method has the disadvantage of burdening the decryptor with too much computing power, and it is hard to predict decryptor's computing power at the time the encryptor encrypts the message.

Trusted decryption agents: In this method, the sender sets a specific date or time. When the time comes, the trusted agent transmits the key to the decryptor for decryption. This method has the advantage of reducing the computing power burden on the decryptor. However, the presence of the agent weakens the security in the sense that the agent can be compromised.

Recently, two research works have been done regarding to this paper. The first is [21], which presents the time-release protocol for the Bitcoin network using the witness encryption. The witness

encryption is the cryptographic primitive where only someone who knows the solution to the specific NP problem can decrypt the encrypted message [22]. The public key in witness encryption is proof-of-work of the Bitcoin network, which is hard-coded. This proof-of-work code is transformed into CNF-SAT using CBMC (bounded model checker for C and C++ programs) and then converted to the subset-sum problem. Because the subset-sum problem is an NP problem, it designs the witness encryption such that the process of solving it is equivalent to the process of obtaining a private key in a witness encryption. This scheme can precisely set the release time by specifying the number of blocks to be mined. However, as the authors say, currently, this scheme is impractical due to astronomical computational overheads.

The second is [23] in which the authors proposed homomorphic time-lock puzzles. To reduce computational overhead or decoding in time-lock puzzles, [23] proposes the method that combines all the puzzles into one puzzle through a homomorphic function and then decodes that puzzle. This method has the advantage of not having to solve all the puzzles. However, it requires the use of homogeneous encryption, which is non-standard and incurs high computational overheads.

3. Time-release Blockchain

This section describes the proposed scheme, time-release blockchain. We have added time-release cryptographic functionalities in the existing blockchain system. In Section 3.1 we describe the structure of the single block and in Section 3.2 we explain the chained blocks structure. In Section 3.3 and Section 3.4, we explain the mining work and time-release techniques.

3.1. Block structure

The header structure of the time-release blockchain is described in Table 3. In the proposed scheme, the fields of the block header are almost identical to those of the bitcoin blockchain except for the following three things. First, nonce is calculated using Equation (3) where SHA256() is the cryptographic one-way hash function [11] and $key_{private}$ is the private key corresponding to the public key in the previous block. In this equation, p is the security parameter of DLP (discrete logarithm problem)-based encryption scheme, i.e., the size of the cyclic group G.

$$SHA256(SHA256(block\_header)) \equiv key_{private} \pmod{p}. \quad (3)$$

Table 3. Structure of the header part in the block of the time-release blockchain.

Size	Field	Description
4 Bytes	Version	Version number for tracking software or for upgrading the protocol
4 Bytes	Timestamp	Approximate time when the block was created (in Unix timestamps).
32 Bytes	Merkle root	Root of the Merkle Tree [8] generated from the transactions in the block
4 Bytes	Difficulty	The Proof-of-Work algorithm difficulty target for this block
32 Bytes	Previous block header hash	The reference to the hash of the previous (parent) block
4 Bytes	Nonce	The value to satisfy Equation (3)
4 Bytes	Public key length (bit)	The number of bits in the public key
Not fixed	Public key	Public key created using the public key of the previous block

Just as in other blockchain schemes with PoW (proof-of-work), the correct nonce value to meet Equation (3) is calculated through many tries. The cryptographic one-way hash function (SHA256()) prevents modification of the block without recalculating the nonce. Since the correct nonce value is congruent to the private key corresponding to the public key in the previous block, this value is automatically released in the new block.

Second, the public key length means the number of bits of the public key (and that of the private key). Third, the public key field is a new public key created using the public key of the previous block where we use the algorithm in Figure 3. The random number generator in Figure 3 uses the standard random number generator such as ANSI X9.31 pseudorandom number generator [24] or RC4 [25] stream cipher with the fixed time value. Therefore, if the same seed value is used, anyone can generate the same random number. Thus, without mining the new blocks, anyone can obtain the public key of the next block using this algorithm.

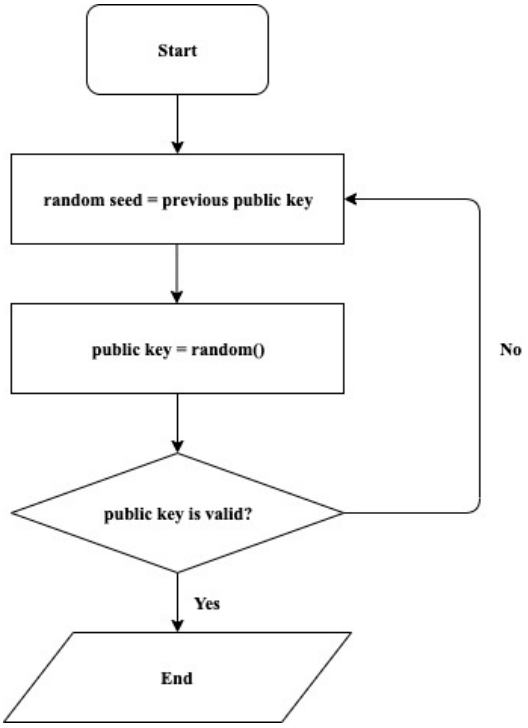

Figure 3. Algorithm to generate the public key.

Figure 4 compares the block structure of the existing Bitcoin blockchain with that of the proposed scheme, which shows that two elements are different from each other: public key and private key length. Even though nonce is included in both, as mentioned previously, PoW of the proposed scheme is different from that of Bitcoin which relies on the cryptographic hash function only.

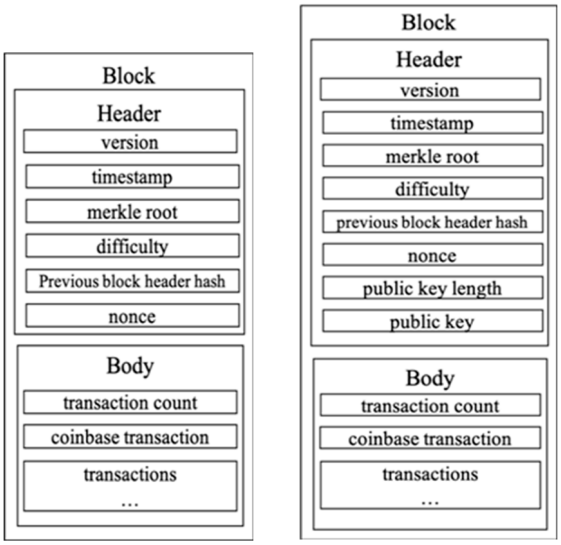

Figure 4. Block structure of Bitcoin blockchain (left) and that of our scheme (right).

Table 4 shows the structure of the body part in the block, which is identical to that of the original Bitcoin.

Table 4. Structure of the body part in the block.

Field	Description
Transaction count	Number of transactions in the block body
Coinbase transaction	The first transaction in a block. It is always created by a miner and it has no inputs. This is the transaction that rewards the miner.
Transactions (tx_1, tx_2, … tx_n)	n transactions in the block

3.2. Chained Block Structure

In our time-release blockchain, a public key in the current block is generated using the public key of the previous block (by the algorithm in Figure 3). The private key for the public key of the current block is calculated using the header information of the next block by Equation (3). Figure 5 shows the chain structure of the time-release blockchain. The one-directional arrow means that the public key of the previous block is used as the seed value for generating the public key of the current block. The bidirectional arrow implies that the public key of the previous block and the private key of the next block (this is calculated by Equation (3)) are paired with each other.

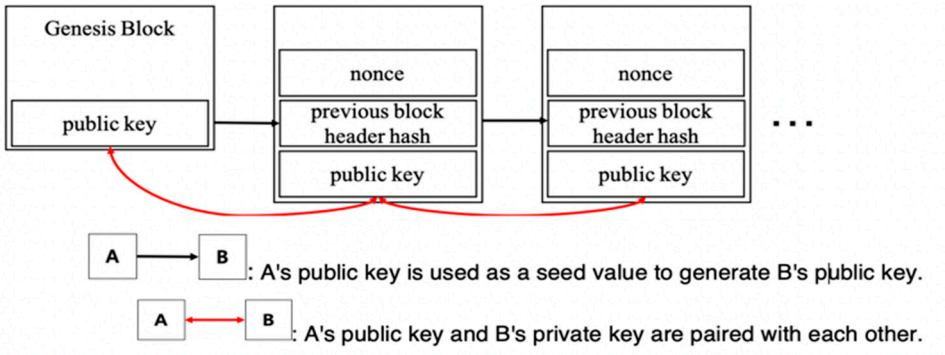

Figure 5. Chained blocks structure of the time-release blockchain.

Figure 6 shows an example of the chained blocks structure in our scheme. In this figure there are 3 blocks (block index: 1, 2, and 3). The public key in the first block, (0x3021a3, 0x578cc, 0x6015bf) is used as a seed value to produce the public key in the second block, (0x3fb4bd, 0xcedf, 0x4a996f). Also, the public key in the first block and the private key calculated using the header information of the second block (=SHA256(SHA256(2nd block_header)) mod p) are paired with each other. Similarly, the second block and the third block have the same relation.

Figure 6. An example of the chained blocks structure.

3.3. Mining a New Block

In our time-release blockchain, by the proof-of-work algorithm, the candidate block becomes a valid block. Mining is done by finding the nonce value to meet Equation (3), which means that the left-hand side of Equation (3) is congruent to the private key (which corresponds to the public key of the previous block). The mining work is shown in Figure 7.

For our time-release blockchain, most of the public key cryptography algorithms that are based on discrete-logarithm problem can be used, e.g., ElGamal encryption or elliptic curve integrated encryption scheme (ECIES).

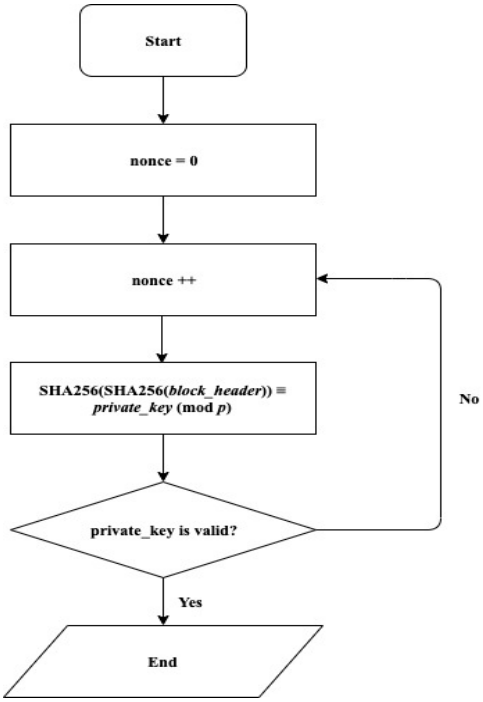

Figure 7. Proof-of-work of the time-release blockchain.

3.4. Time-release Technique

By using our time-release blockchain, users can freely encrypt the plaintext to produce the ciphertext: the plaintext is encrypted with the public key in the specific block, e.g., the recently released block. When the next block of this block is mined, corresponding private key is released using Equation (3) and the ciphertext can be decrypted. Suppose that a new block is created every t minutes on average. If we encrypt the plaintext using the public key in the current block, t minutes later, a new block is mined and the corresponding private key can be calculated by using the header information of the new block. This means that the ciphertext can be decrypted by anybody, i.e., sealed information is released in t minutes later.

How can we seal information such that it will be released in $k \times t$ minutes? Note that the public keys are calculated using the algorithm in Figure 3. This algorithm shows that the public keys in next k blocks can be easily calculated without the mining work. The first thing to do is to prepare k public keys using the algorithm in Figure 3. Then, we encrypt the plaintext multiple times using these public keys. Figure 8 shows the ciphertext encrypted with k public keys. In this figure, C_k means the ciphertext that is produced from C_{k-1} using the k-th public key (where C_0 is the plaintext).

All PoW-based blockchains are autonomously adjusted such that a new block is mined at regular intervals. Therefore, even if the number of nodes changes or the computing power of the nodes changes over time, new blocks are mined at constant intervals. Because the proposed time-release blockchain uses this property, it does not have the shortcoming of the time-lock puzzles schemes (which are hard to predict the precise release-time).

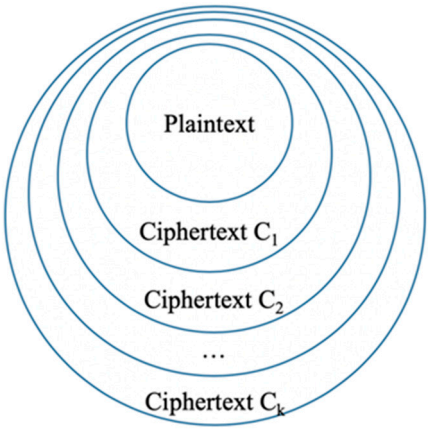

Figure 8. Ciphertext that is encrypted with k public keys.

4. Prototype Implementation of the Time-release Blockchain

This section describes the prototype implementation of our time-release blockchain which is described in Section 3. We implemented using Python 3.7.3 where the source can be accessed at https://github.com/Sangoou.

4.1. Block Structure

Figure 9 shows class definition regarding to the header of the block. The variable self.difficult means degree of difficulty to find the correct nonce. In our protocol we adjust difficulty by changing the size of the public/private key, public_key_size.

The body part is not implemented in our prototype because the proposed scheme and the conventional blockchains are the same as each other.

```
23 class Block:
24     def __init__(self, index, timestamp, data, next_public, public_key_size, nonce=None, before_header_hash = None):
25         self.version = 1.0
26         self.index = index                                          #block index
27         self.timestamp = timestamp
28         self.data = data
29         self.difficult = next_public.p      #difficulty when calcuating the nonce
30         self.before_header_hash = before_header_hash            #header hash of the previous block
31         self.nonce = nonce
32         self.public_key_size = public_key_size              #length of the public key
33         self.next_public = next_public                              #public key value
```

Figure 9. Class definition of the header of the block.

4.2. Creating a Genesis Block

Figure 10 shows the method for creating a genesis block. The function elgamal.generate_keys() returns the public key object. In Line 48, create_genesis_block() returns the genesis block.

```
46 def create_genesis_block():
47       cipher = elgamal.generate_keys(seed=0xfffffffffffff,iNumBits=18)
48       return Block(0, time.time(), {"transactions": None}, cipher, 18)
```

Figure 10. Implementation of creating a genesis block.

4.3. Proof-of-Work (PoW)

Figure 11 shows the mining part of our time-release blockchain. The first parameter, last_block is the reference variable corresponding to the previous block. Lines 81 to 98 are related with the implementation of the algorithm in Figure 7. The variabl is a candidate value to be used as a nonce.

From this value, the left-hand side of Equation (3) is calculated. By applying mod p operation, we can get the candidate private key. As shown in line 89, we use ElGamal cryptography for implementation. If the candidate private key is correct with respect to the previous block's public key, mining is successful, and the function returns the corresponding block. Lines 94 to 98 are the implementation of selecting the longest chain, which belongs to the conventional blockchain systems.

```
81 def proof_of_work(last_block, candidate_block, blockchain):
82     i=0
83     start_time = time.time()
84     while True:
85         candidate_block.nonce = i
86
87         hash_header = int(candidate_block.hash_header(),last_block.public_key_size) % (last_block.difficult-1) + 1
88
89         if last_block.next_public.h == elgamal.modexp(last_block.next_public.g, hash_header ,last_block.next_public.p):
90             break
91
92         if (time.time()-start_time) > 30:
93             start_time = time.time()
94             new_blockchain = consensus(blockchain)
95             if new_blockchain:
96                 return False, new_blockchain
97         i = i+1
98     return candidate_block, blockchain
```

Figure 11. Implementation of proof-of-work.

4.4. Adjusting Difficulty

Figure 12 shows the implementation of difficulty adjustment in our time-release blockchain. The proposed method uses the similar difficulty adjustment algorithm as in the conventional blockchain systems. This adjusting function is called whenever a specific number (N) of blocks (N = 100 in our code) are mined and the length of the public key of the block header is adjusted by comparing the desired time period with the actual elapsed time for mining N blocks.

```
60 def Calculate_Difficulty( difficulty ):
61     global flag_, start_time, time_
62
63     if flag_ == 0 :
64         start_time = time.time()
65         flag_ = 1
66
67     else :
68         if time.time() - start_time > time_ :
69             difficulty = difficulty - 1
70
71         elif time.time() - start_time < time_ :
72             difficulty = difficulty + 1
73
74         start_time = time.time()
75
76     return difficulty
```

Figure 12. Implementation of difficulty adjustment.

5. Application: E-voting with the Time-release Blockchain

This section shows a practical example for using our time-release blockchain: an e-voting system. This voting protocol assumes that the time-release blockchain supports smart contracts just as in Ethereum [26]. This protocol can be considered an enhancement of [17], where [17] relies on the access control (by the administrator) to provide privacy. First, we briefly describe [17] and then explain how to provide privacy using time-release blockchain. It has three components, which are as follows.

- Smart voting contracts: contracts that run on the blockchain. They handle registration, election process, and recording transactions.
- Administrator: It creates the election, registers voters, decides the lifetime of the election and setups the blockchain.

- Voter: An individual who is eligible to vote. After authentication, the voter loads the election ballot, casts the vote, and verifies the vote after the election is over.

Figure 13 shows the overall e-voting process. Each step is described as follows. (For further detailed explanation, refer to [17]).

1) Election creation: The administrator creates a new election.
2) Voter registration: After an election is created, the administrator defines a list of eligible voters. Then, each voter contacts the administrator for authentication. The administrator can distribute voting rights through blind signatures [19] (for protecting privacy). After authentication is finished, a unique wallet is generated for each voter.
3) Voting transaction: Each voter interacts with the ballot smart contract. The voter sends his/her voted ballot the contractor. After receiving this, the contractor generates a unique transaction ID and makes a transaction containing the transaction ID and the ballot. Then, the contractor uploads the transaction in the blockchain while it sends the transaction ID back to the voter. Note that the blockchain network is not open to the voters during the voting procedure.
4) Tallying the results: Tallying can be done on the fly in the smart contract because it does not use any encryption for the ballot.
5) Verification: After the election is done, voters can access the blockchain network. In the network, all transactions have been recorded. With the transaction ID, voter can check that his/her vote has not been modified. Also, all transactions are not encrypted, anybody can check the integrity of tallying work.

Note that the above protocol has shortcomings in that to keep privacy during the voting procedure, users are not allowed to access the blockchain, where the access control is handled by the administer. Also, the administrator/smart contract can access the blockchain during the voting procedure, which may raise privacy concerns

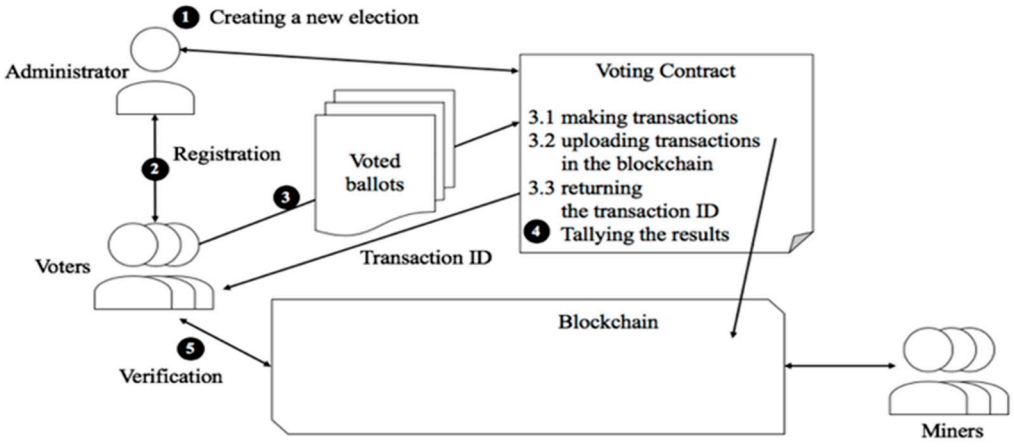

Figure 13. The overall procedure in e-voting using blockchains [17].

This protocol can be enhanced using the time-release blockchain as follows. Figure 14 shows this protocol.

1) Election creation: The same as that of the original protocol.
2) Voter registration: The same as that of the original protocol.
3) Voting transaction: Each voter encrypts his/her voted ballot using the public keys in the time-release blockchain such that confidentiality is kept until the deadline. Then, he/she interacts with the ballot smart contract. The voter sends the encrypted voted ballot the contractor. After receiving this, the contractor generates a unique transaction ID and makes a transaction containing the transaction ID and the ballot. Then, the contractor uploads the transaction in the blockchain while it sends the transaction ID back to the voter. Note that the encrypted ballots are not open during the voting procedure due to time-release encryption.

4) Tallying the results: After the deadline expires, (corresponding private keys have been released and) all encrypted ballots can be decrypted. Tallying is done in the smart contract.
5) Verification: After election is done (the deadline expires), voters can access the blockchain network. In the network, all transactions have been recorded and decrypted. With the transaction ID, the voter can check that his/her vote has not been modified. Also, because all transactions can be decrypted, anybody can check the integrity of tallying work.

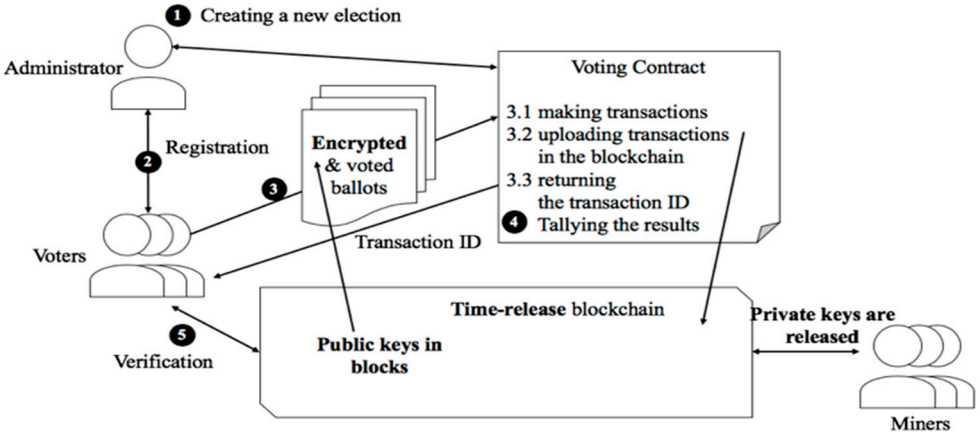

Figure 14. Proposed e-voting procedure that is based on the time-release blockchain [17].

6. Security Analysis

This section analyzes the security of the proposed scheme. Section 6.1 analyzes the security of the proof-of-work algorithm of the time-release blockchain and then we compare the security issues with the previous work in Section 6.2.

6.1. Security Analysis of the Time-release Blockchain

The original blockchain system is designed to provide very strong security such that the attackers cannot forge/modify the contents of the block provided that the attacker's computing power does not exceed a half of all participants' computing power in the network. In our time-release blockchain, in order to modify the content of the existing block, the attackers should find the collision of the cryptographic one-way hash function in Equation (3), which is considered impractical. Section 1 deals with this type of attack. For the attack for finding the private key without using Equation (3), the attack can get some advantages over other nodes. Our system is designed secure under assumption that first, attacker's powers are significantly lower than the whole computing power, and second, we provide the expected time interval where the earliest time means the lower bound time when attackers can find the private key and the latest one implies the expected time when the private key is released in the normal protocol (using Equation (3)).

6.1.1. Attack for Modifying the Existing Block

Assume that there is an attacker who wants to modify the valid block in the chain. If the body in the block is modified, the root of the Merkle tree [8] will be changed and the equality in Equation (3) will fail with the extremely high probability (=1/p where p is the security parameter of DLP-based encryption scheme). The attacker can change the nonce value to meet Equation (3), but this work will take almost the same as finding a new valid block.

6.1.2. Attack for Finding the Private Key of the Block

Assume that the attacker tries to compute the private key without using Equation (3). First, suppose that he/she uses the brute-force attack for finding the key. For this approach, the attacker

has a slightly higher probability of success compared with the normal nodes using Equation (3) because the attacker can rule out already tried candidates for the key. Precisely speaking, for n times tries, the attacker's success rate is $1-(p-1)/p\times(p-2)/(p-1)\times\ldots\times(p-n)/(p-n+1) = 1-(p-n)/p$. For the normal node using Equation (3), the success rate for n tries is $1-((p-1)/p)^n$. For example, when p = 2,147,483,647, Figure 15 shows the probability for finding the key for the brute force attack vs. the probability for finding the key using Equation (3). In this graph, x-axis means the number of trials while the y-axis represents the success probability. This graph shows that brute force attack is slightly more efficient than finding the key using Equation (3).

Second, attackers can use efficient algorithms for finding private keys. If we use elliptic curve cryptography (ECC) encryption, the most efficient algorithms for solving ECDLP problems are the Pohlig–Hellman and Pollard–Rho algorithms [27]. With careful selection of the group in ECC ciphers, the time complexity of both algorithms is $O(e^{0.5n})$ [27], where n is the key length in bits. This shows that the attacks that directly find private keys are much more efficient than the mining work.

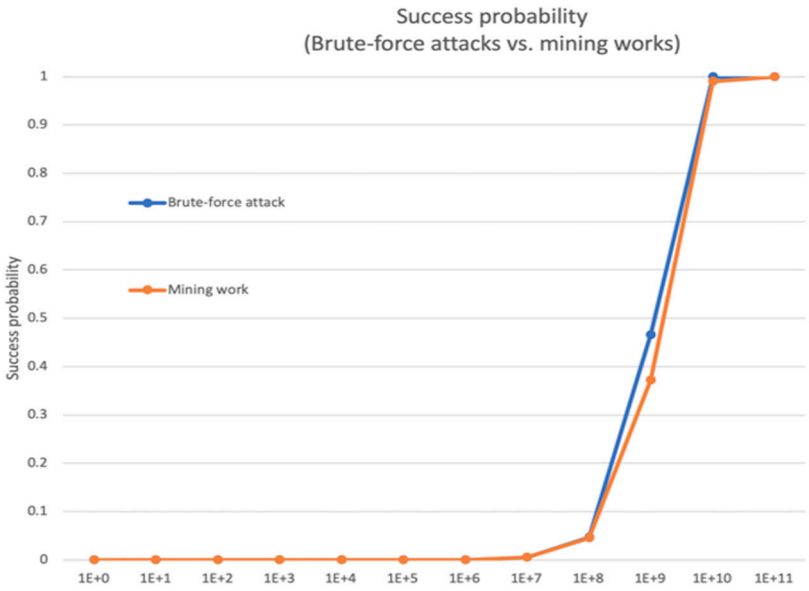

Figure 15. Brute-force attack vs. mining works.

However, if the attack fails, mining rewards cannot be obtained at all, so the number of attackers will be relatively few compared to the whole network nodes. Currently, the sum of the computing power of the nodes participating in Bitcoin is now over 100 quintillion hashes per second (EH/s) [28], which is about 2^{48} times more than a single PC (having about 300 hashes per second [29]).

As an example, suppose that we use 60-bit ECC algorithm. The attacker must perform the operations of $O(e^{30})$ using the Pohlig–Hellman or Pollard–Rho algorithm. On the other hand, the blockchain network having 248 times more powerful than the attacker should compute the work of $O(2^{60})$. This shows that the network is much more likely to find the key than the attacker.

In summary, if an attacker uses the efficient algorithms for directly searching the private key, it can run the algorithms that are more efficient than the mining work. However, the computing power of the entire block chain is much more powerful, our time-release blockchain is secure, provided that there are not so many attackers exist.

To defend against this kind of attack, we can secure the system by increasing the number of layers (described in Section 3.4). We provide the expected time interval (T_1, T_2) as follows. T_2 denotes the average time for mining a single block and the expected time for the attacker to find the private key is T_1 ($T_1 < T_2$). The user can use k layers and the expected release time will be from $k \times T_1$ to $k \times T_2$.

6.2. Comparison with the Previous Work

6.2.1. Time-release Cryptography vs. Time-release Blockchain

Compared with trusted decryption agents (explained in Section 1), the proposed method is much simpler and does not suffer from trust-related problems with the agents since it does not require any agents.

In time-lock puzzles, it is hard to predict the computing power of the recipient, i.e., the time to decrypt the ciphertext. On the other hand, the proposed scheme continuously predicts and adjusts the mining time of the blockchain network nodes, which can cope with environments where the computing power has been changing unceasingly. As seen in Table 5, the proposed scheme is very practical compared with the previous works.

Table 5. Comparison of the previous time-release cryptography and our time-release blockchain.

	Time-release Cryptography [30]	**Time release Cryptography + Witness Encryption [21]**	**Time-release Cryptography + Homomorphic cryptography [23]**	**Our scheme**
Computation overhead	Very large	Practically infeasible	Very large	Efficient
Decoding-time predictability	Difficult	Easy	Difficult	Easy
Responding to the changed computing power over time	Impossible	Possible	Impossible	Possible
Whether to use the standard encryption or not	X	X	X	O

6.2.2. Comparison with Blockchain-based E-vote protocols

In Section 5, we have proposed the e-voting protocol that uses the time-release blockchain. Our protocol distributes voting rights through blind signatures [19]. This allows only those who have the right to vote (eligibility) while protecting privacy. Time-release cryptography hides the identity of the voters and ballots (anonymity). It allows the person to resubmit their votes at any time during the voting period (forgiveness). Because our protocol supports both forgiveness and anonymity, coercion resistance is automatically supported.

Integrity and verifiability of voting are inherited from the characteristics of the original blockchain. Also, after voting has begun, everyone has the same authority, and no one can invalidate the vote (i.e., admin freedom), which is different from other previous voting protocols.

Table 6 shows the comparison result in between the previous e-voting protocols using the blockchain and our protocol using the proposed time-release blockchain. Our protocol meets all the requirements (except for physical verification). Note that physical verification is related to offline identification and is a common problem in all e-voting protocols [20].

Table 6. Comparison of the previous e-voting protocols and our scheme.

	Polys	**BroncoVote**	**RankedChoice**	**BitCongress**	**Follow my vote**	**Proposed scheme**
Eligibility	O	O	O	X	O	O
Anonymity	O	O	O	O	O	O
Verifiability	O	O	O	O	O	O
Integrity	O	O	O	O	O	O

Physical verification	-	X	X	-	-	-
Forgiveness	-	-	O	O	O	O
Admin freeness	X	X	X	X	X	O

- : Undetermined due to lack of detailed information, O: Supported, X: Unsupported

7. Conclusions

In this paper, we have presented a new practical time-release blockchain scheme. In order to solve the shortcomings of the time-lock puzzle, we have merged the proof-of-work in the blockchain system and time-lock puzzle algorithm. Our scheme provides an efficient way to publish content at a specific time by automatically finding the key (to decrypt the content) from the blockchain system. To estimate the feasibility of our scheme, we have implemented a prototype version of our blockchain system using Python, which shows that the proposed scheme works properly. Also, we give a practical example for using our scheme, namely the smart contract e-voting system. The proposed method has the following advantages. First, by linking the difficulty of the blockchain and the difficulty of the time-lock puzzle, the decryption time is automatically adjusted even if the miner's computing power changes over time. Second, unlike previous time-lock puzzle schemes, our algorithm does not require additional computation work for solving the puzzle. Third, unlike the trusted decryption agent schemes, our scheme does not need any trusted third parties. Fourth, the proposed method uses standard cryptographic algorithms, whereas the previous work relies on very complex non-standard algorithms.

Author Contributions: Methodology, S.-W.C.; Software, J.-I.K.; Writing – original draft, Y.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2017R1D1A1B03029550).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cheon, J.H.; Hopper, N.; Kim, Y.; Osipkov, I. Provably Secure Timed-Release Public Key Encryption. *ACM Trans. Inform. Syst. Secur.* **2008**, *11*, 1–44.
2. Rivest, R.L.; Shamir, A.; Wagner, D.A. Time-Lock Puzzles and Timed-Release Crypto, Massachusetts Institute of Technology, White Paper, 1996. Available online: https://people.csail.mit.edu/rivest/pubs/RSW96.pdf (accessed on 29 March 2020).
3. Zheng, W.; Zheng, Z.; Chen, Z.; Dai, K.; Li, P.; Chen, R. NutBaaS: A Blockchain-as-a-Service Platform. IEEE *Access* **2019**, *7*, 134422–134433.
4. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System, Satoshi Nakamoto Institute, White paper, 31 October 2008. Available online: https://git.dhimmel.com/bitcoin-whitepaper/ (accessed on 29 March 2020).
5. Syed, T.A.; Alzahrani, A.; Jan, S.; Siddiqui, M.S.; Nadeem, A.; Alghamdi, T. A Comparative Analysis of Blockchain Architecture and its Applications: Problems and Recommendations. IEEE *Access* **2019**, *7*, 176838–176869.
6. Yang, W.; Aghasian, E.; Garg, S.; Herbert, D.; Disiuta, L.; Kang, B. A Survey on Blockchain-Based Internet Service Architecture: Requirements, Challenges, Trends, and Future. IEEE *Access* **2019**, *7*, 75845–75872.
7. Cryptoticker, What is the Blockchain Data Structure? Available online: Available online: https://cryptoticker.io/en/blockchain-data-structure/ (accessed on 29 March 2020).
8. Merkle, R.C. A Digital Signature Based on a Conventional Encryption Function. *Adv. Cryptol.—CRYPTO '87. Lecture Notes Comput. Sci.* **1988**, *293*, 369–378.
9. Wang, W.; Hoang, D.T.; Hu, P.; Xiong, Z.; Niyato, D.; Wang, P.; Wen, Y.; Kim, D.I. A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks. *IEEE Access* **2019**, *7*, 22328–22370.

10. Liu, Z.; Luong, N.C.; Wang, W.; Niyato, D.; Wang, P.; Liang, Y.-C.; Kim, D.I. A Survey on Blockchain: A Game Theoretical Perspective. *IEEE Access* **2019**, *7*, 47615–47643.
11. Courtois, N.T.; Grajek, M.; Naik, R. Optimizing SHA256 in Bitcoin Mining. In Proceedings of CCS 2014, Scottsdale, AZ, USA, 3–7 November 2014; pp. 131–144.
12. Antonopolulos, A.M. *Mastering Bitcoin,* 2nd ed.; O'Reilly Media: Sebastopol, CA, USA, 2017.
13. Alyoshkin, R. Polys–Online Voting System Whitepaper. Available online: https://docs.polys.me/technology-whitepaper (accessed on 29 March 2020).
14. Dagher, G.; Marella, P.B.; Milojkovic, M.; Mohler, J. BroncoVote: Secure Voting System using Ethereum's Blockchain. In Proceedings of ICISSP 2018, Funchal, Madeira-Portugal, 22–24 January 2018; pp. 96–107.
15. Yang, X.; Yi, X.; Nepal, S.; Kelarev, A.; Han, F. A Secure Verifiable Ranked Choice Online Voting System Based on Homomorphic Encryption. *IEEE Access* 2018, 99, doi 10.1109/ACCESS.2018.2817518.
16. Hardwick, F.S.; Hioulis, A.; Akram, R.N.; Markantonakis, K. E-Voting with Blockchain: An E-Voting Protocol with Decentralisation and Voter Privacy. *IEEE* **2018**, arXiv:1805.10258.
17. Hjálmarsson, F.Þ.; Hreiðarsson, G.K. Blockchain-Based E-Voting System. In Proceedings of 2018 IEEE 11th International Conference on Cloud Computing, Seattle, WA, USA, 2–7 July 2018, pp. 983–986.
18. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613.
19. Chaum, D. Blind signatures for untraceable payments. In Proceedings of Advances in Cryptology 1983, Santa Barbara, CA, USA, 21–24 August 1983; pp. 199–203.
20. Sayyad, S.F.; Pawar, M.; Patil, A.; Pathare, V. Features of Blockchain Voting: A Survey. *Int. J. Innov. Res. Sci. Technol.* **2019**, *5*, 12–14.
21. Liu, J.; Garcia, F.; Ryan, M. Time-release Protocol from Bitcoin and Witness Encryption. *IACR Cryptol. ePrint Arch.* **2015**, *482*, 2015.
22. Garg, S.; Gentry, C.; Sahai, A.; Waters, B. Witness Encryption and its Applications. *IACR Cryptol. ePrint Arch.* **2013**, *258*, 1–30.
23. Malavolta, G.; Thyagarajan, S.A.K. Homomorphic Time-Lock Puzzles and Applications. In Proceedings of Crypto 2019, Santa Barbara, CA, USA, 18–22 August 2019; pp. 620–649.
24. Keller, S.S. NIST-Recommended Random Number Generator Based on ANSI X9.31: Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms. In *NIST Information Technology Laboratory-Computer Security Division, National Institute of Standards and Technology*; ANSI X9.31; NIST: Gaithersburg, MD, USA, 2005.
25. Singhal, N.; Raina, J.P.S. Comparative Analysis of AES and RC4 Algorithms for better utilization. *Int. J. Comput. Trends Technol.* **2011**, *2*, 177–181.
26. Chinchilla, C. Ethereum–A Next-Generation Smart Contract and Decentralized Application Platform, White paper. Available online: https://github.com/ethereum/wiki/wiki/White-Paper (accessed on 29 March 2020).
27. Sommerseth, M.L.; Hoeiland, H. Pohlig-Hellman Applied in Elliptic Curve Cryptography, Technical Report, University of California Santa Barbara, 2015. Available online: https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Sommerseth+Hoeiland.pdf (accessed on 29 March 2020).
28. Suberg, W. Bitcoin Hash Rate to Hit a Milestone 100 Quintillion for the First Time," Cointelegraphy, Sepember 2019. Available online: https://cointelegraph.com/news/bitcoin-hash-rate-to-hit-a-whopping-100-quintillion-for-the-first-time (accessed on 29 March 2020).
29. Hashrates, M. Intel Core i7-6700 Processor Hashrate, 2019. Available online: https://www.minershashrates.com/intel-i7-6700-processor-hashrate/ (accessed on 29 March 2020).
30. Mao, W. Time-Release Cryptography, In Proceedings of Selected Areas in Cryptography'01 2001, 342-357.