A Fully Decentralized Time-Lock Encryption System on Blockchain

Wei-Jr Lai
National Taiwan University
r05922108@cmlab.csie.ntu.edu.tw

Chih-Wen Hsueh
National Taiwan University
cwhsueh@csie.ntu.edu.tw

Ja-Ling Wu
National Taiwan University
wjl@cmlab.csie.ntu.edu.tw

Abstract— To make a time capsule on the Internet, which will be opened at a planned time in the future, without third parties' involvement has always been a difficult problem. Although there are many researches worked on various time-lock systems, they may have some shortcomings like uncertainty in decryption time, not fully decentralized, hard to estimate the required computing resources. In this paper, we proposed a protocol and a reliable encryption scheme to make time-sensitive message be opened on time at a fully decentralized environment, which is then integrated with the blockchain to adapt to different computing power situations. The method also provides the capability of incorporating with appropriate incentives for encouraging participants to contribute their computing resources, which makes our system more suitable for real world applications.

keyword: Time-Lock Encryption, Proof of Work, Privacy Preserving, Blockchain

I. Introduction

There are many kinds of time-critical sensitive information whose degree of sensitivity or importance will be reduced (or even zeroed out) after a specific period of time. Let's look at some examples, in the following, appeared in our practical life first. In an on-line auction system, bidders wrote down their bids independently and this information shouldn't be opened until the end of the auction. Currently, online auctions rely on a middleman to record the bids of each participant and reveal the winner once the bidding time is up. Similarly, in an E-voting system, the information on ballots shouldn't go public until the tally phase has started. In general, the ballots are issued and tallied up by a trusted third party (such as the government or the administrator).

In contrast to traditional symmetric and asymmetric cryptosystems [1], whose managing strategies concern mainly about who can encrypt and decrypt a message with proper keys, the cryptography system needed to support the above-mentioned two examples has an inherently different goal: providing a guarantee about "at what time" one can decrypt the encrypted message. That is, the encrypted message should not be decrypted until a specific time instance is reached. Academically, we named this kind of cryptosystems the "Time-Lock", "Timed-release", or "Time-Lapse" Cryptographic systems [2,3,4,5].

Intuitively, one method for encrypting such a time critical message is to rely on a trusted third party (TTP) [5], which reveals decryption keys at a pre-negotiated time. In other words, the voting security relies crucially on the assumption that the TTP is trustworthy. In particular, the TTP must not use its ability to perform ciphertext decryption earlier than the declared time or date, per voters' request in any malicious way. To avoid single point of failure, some constructions share the decryption key among several (trusted) parties. However, a collusion of some (or in the worst case all) parties against the encrypter will always be able to decrypt the ciphertext before the deadline set in TTP-based approaches. Even worse, a misbehaved third party or dishonest middleman may launch a man-in-the-middle attack, which is very hard to resist. Clearly, if any one of the parties chooses to defect and release the key before the planned time, the encrypted information will be publicly available before the planned schedule. Depending on the contents of the information, this could have potentially disastrous consequences.

The other line of research [2,3,4] considers constructions that require the receiver of a ciphertext to perform a feasible, but computationally expensive exploration of a decryption key. This puts a considerable computational overhead on the receiver. For example, assume a sender wants the voting system to guarantee that the ciphertext generated by him must not be decrypted within, say, one week. In order to meet this requirement, the sender would have to be able to estimate the computational resources available to the receiver rather accurately; and at the same time, the receiver would have to dedicate all these computational resources for one week to the task of key exploration. The main challenges faced in this line of research are situations where the ciphertext is made public and there are many different receivers. For instance, suppose the ciphertext is posted on the Internet, and everybody should be able to decrypt it immediately after the deadline. Since there are many receivers with different computational resources, it is very difficult for the sender to design a key exploration process which matches both the requirements of computational resources and the given deadline, for all of the receivers, simultaneously. Therefore, [6] commented that 'It seems impossible to encrypt with any known timed-release encryption scheme in a way, such that all receivers are able to decrypt at the same time, unless one relies on TTPs'.

Moreover, even if all the involved parties are able to rather accurately solve the computational problem within a given time slot, this kind of approaches would further require that all parties begin their computations at the same time and all parties are able and willing to put all their resources to explore the decryption key. We think it an interesting question in its own to ask 'Is it possible to avoid or at least somewhat relieve the above-mentioned requirements?'

For the ease of explanation, let's go back to the on-line bidding example, again. To avoid the single point of failure (or the downside of a misbehaved middleman), a distributed online auction system would allow each user to encrypt and submit his bid to the other users participating in the auction and each user could begin decrypting the bids of the other users to independently verify the winner. If every user encrypted their bid to withstand at least the time remaining in the auction when they submitted it, no participant could determine any other user's bid before the auction ended.

Interestingly, notable similar scenarios and requirements occurred in recent development of Bitcoins and other cryptocurrencies, which are based on the possession of information rather than the possession of physical materials. A variety of distributed and time-sensitive schemes have been proposed which make the reality of encrypting money or transactions, for a later usage, becomes possible. In fact, cryptocurrencies relied on the blockchain technology which provides an immutable record of all confirmed transactions (each of which has been signed with the private key of the user generating them). Third parties perform a process called mining involves confirming transactions which have occurred and add them to the end of the blockchain, and which is incentivized by possible rewards for confirming transactions.

In facing of similar time-lock requirement, i.e. the safe concatenation of the confirmed transactions, before a certain amount of time, is guaranteed by using the well-known Proof-of-Work (PoW) scheme, where a certain amount of computational time is required to mine a coin (i.e., decrypt a message). Inspired by this idea, our work focuses on developing a fully decentralized time-lock encryption scheme which does not rely on any trusted authority to function.

II. Related Work

As pointed out in [4], time lock encryption allows users to send a message "to the future". The key characteristics of time-lock encryption include the following, which should be fulfilled *simultaneously*:

- Decryption is *non-interactive*. That is, the message sender, say Alice, is not required to be available for decryption.
- Time-lock encryption does not rely on TTPs. Thus, Alice is not required to trust any of the third parties for keeping (or sharing) decryption keys in secret until the deadline has passed.
- Parties interested in decrypting a ciphertext are not forced to perform expensive computations until the decryption succeeds. This means that a party which simply waits till the deadline for starting decryption has passed will be able to decrypt the ciphertext at about the same time as any of the other parties who attempts to decrypt the ciphertext earlier by performing a large (but reasonably bounded) number of computations. In other words, all reasonably bounded parties will be able to decrypt a ciphertext at essentially the same time, regardless the computational resources they have.

The fact that these features have to be achieved simultaneously makes time-lock encryption a fascinating cryptographic primitive, which enables applications that seem impossible to achieve with classical encryption schemes.

Reference [6] proposed two different ways to create a computational load gap between the key generation and the key crack. Someone who wants to make time capsules in the cyberspace can encrypt his messages with the time specific key he generated. Anyone who wants to decrypt the messages have to do huge calculations. The computational load gaps of the two methods are respectively $O(\log n)$ and $O(\sqrt{n})$ in sequential realization, and can both be reduced to $O(n)$ if parallel realization is adopted. If there are lots of participants who want to encrypt time-sensitive messages, each one of them has to create a puzzle for himself. And every message has to be cracked individually.

In this work, a decentralized key generation method which enables participants to encrypt their time-sensitive messages with same public key, is proposed. Most interestingly, in our approach, only one process is needed to decrypt all messages. The key generation for encrypting messages take only $O(1)$ computational load and the whole processes can be built in a trustless environment.

III. Proposed Method

3.1. A Naïve Time-Lock Encryption Scheme and Its Shortages

A naive way to generate a public key without concerning the corresponding private key can be achieved via Diffie-Hellman key exchange (DHKE) algorithm. Different from general scenarios, we use DHKE to build an asymmetric cryptography system, that is, on a chosen elliptic curve, Alice generates $\boldsymbol{xG}$, Bob generates $\boldsymbol{yG}$, and both of them open $\boldsymbol{xyG}$ to the public, where $\boldsymbol{G}$ is a base point of the elliptic curve. Now, participants can encrypt their sensitive messages with this public key $\boldsymbol{xyG}$. Of course, we can extend the number of partial secret holders (Alice and Bob, in this case) from two to many as [7], to make the process more distributed. This approach allows us to build a public key for encrypting secrets without considering the corresponding private key. But this method might have the following two problems:

1. Once all of the partial key holders collude together, the sensitive messages will be publicly aware before the scheduled time.
2. Once one of the partial key holders does not open or loses his secret, the encrypted messages will never be opened again.

Of course, we can somewhat relieve the problem 2 by integrating the method with a deposit scheme to encourage secret holders more willing to follow the protocol. That is, secret holders have to deposit an appropriate amount of cryptocurrency in advance, and they can get their deposits back if they open their secret correctly. However, how to determine the proper amount of deposits so that problem 2 can be migrated smoothly is another difficult problem. Anyway, there is still no effective way to effectively prevent the participants from colluding with others to open sensitive data earlier.

3.2. A Novel Time-lock Encryption Scheme with a Common Decryption Key

For building a decentralized time lock encryption scheme within a completely untrusted environment, first in this section, a Common-Key scheme is proposed to prevent partial key holders from having any key related information advantage over other participants.

For building a fully decentralized time-lock encryption scheme and excluding any involvement of key managers, the following procedures must be considered thoroughly.

First, we have to find an elliptic curve with appropriate order, which means the discrete logarithm problem (DLP) on the curve mustn't be computationally infeasible; in other words, the DLP would be compromisable within an expectable time. However, for security and privacy protection reason, the DLP on the chosen curve should not be compromised immediately. An elliptic curve possesses the pre-described characteristics is called a Semi-Feasible Elliptic Curve (SFEC), in this work. We can select appropriate prime numbers to build a not-so-secure cryptosystem, based on an SFEC, with the techniques presented in [8]. Of course, the expected compromised time of the cryptosystem could be set according to the decryption related parameters. That is, we can choose different elliptic curves for building time-lock encryption schemes with different expected compromised time.

Second, we need a hash function $\boldsymbol{H_e}$ mapping a binary string with arbitrary length to a point on the predefined SFEC [9]. That is,

$\boldsymbol{H_e}\colon \{0,1\}^* \rightarrow \boldsymbol{E}(\boldsymbol{F_q})$, where $\boldsymbol{F_q}$ is the integer field modulo a prime $\boldsymbol{q}$ and $\boldsymbol{E}$ is the selected SFEC.

Now, given a random input to $\boldsymbol{H_e}$, we can obtain the coordinate of a point on SFEC and it is treated as the public key of our time lock encryption scheme. Notice that, under this construction, no one has any pre-knowledge about the corresponding private key within a certain period of time because the process of generating the public key doesn't rely on the corresponding private key, which implies the above key-pair can be used to encrypt sensitive data within the preset time period. Since, as pre-described, DLP on the selected SFEC is compromisable, the singular party who designated the input of the hash function is able to calculate the private key with respect to the output of $\boldsymbol{H_e}$. In other words, he can eavesdrop the encrypted messages when the messages are not gone public yet. In order to prevent this shortage, we presume the input of the hash function is changeable by any of the participants, before the encryption key-pair is fixed. That is, any participant can publish his or her chosen key-related sequence onto an Ethereum smart contract within the setup time. After that, a function in the contract will concatenate all of the involved sequences, orderly, as an input to the hash function $\boldsymbol{H_e}$ and the hash output (i.e., the encryption public key) can then be used to generate the corresponding common private key. Since each participant (voter) contributes only part of the concatenated input, no one knows the encryption public key in advance, and therefore, the common private key before the end of the pre-defined period of time. Voters can then encrypt their sensitive messages with the so-obtained public key. Since nobody can pre-determine the input of the hash function, we can ensure that DLP can only be correctly solved after the common private key has been compromised.

Third, we start to solve DLP on the chosen SFEC. The complexity of the well-known Pollard's rho attack [10] to this problem is approximately $O(\sqrt{n})$, where n is the order of the elliptic curve, and it can be sped up by parallelizing the computation. Therefore, we estimate the required computing power for solving DLP on SFEC, within a particular period of time, according to the order of the chosen SFEC. As long as the difference between the designated compromised time of the above-mentioned SFEC-based key generation scheme and the scheduled release time is well-designed, attackers are hard to eavesdrop ballot related information before the end of the scheduled time even though they had huge computing resources. The privacy of encrypted messages will be invaded only when the common private key is cracked before the scheduled release time.

The whole system can be further simplified as follows.

Step-1 Any one of the participants can select a random number and publish it on an immutable bulletin board, which is referring to the Ethereum blockchain in this work.

Step-2 An Ethereum-based smart contract concatenates all the numbers published in (Step-1) together, as an input to $\boldsymbol{H_e}$, and obtains a point on SFEC.

Step-3 Any one, on the blockchain, can then try to solve DLP on the chosen SFEC and publish the corresponding private key on the Ethereum smart contract, with which other voters are able to decrypt the encrypted ballots.

Under such a construction, no one knows the exact input to the hash function until (Step-2), because any one of the participant can change his or her choice in (Step-1). In other words, participants encrypt messages with a composite public key such that no one can obtain the corresponding common private key, required in (Step-2), beforehand. As pre-described, with the selection of an elliptic curve with proper order, we can generate a public key such that the corresponding private key is expected to be found within a particular period of time. Therefore, no one has more information, or privilege, than others during the entire election processes.

In summary, if we set the time capsules that can only be opened after the scheduled release time, then all the participants needn't to worry about the key managing issue and no participants are able to reveal their secrets or collude together to break the security and/or invade the privacy of the proposed time-lock system.

IV. Experiment and discussion

We ran Pollard's rho attack on the pre-built elliptic curves with different orders. The tested bits of prime of elliptic curves include 32, 36, 40, 44, 48, and 52. Since different computers are used for measuring the time consumption, for fairness, we record the step-cost (i.e., the number of executed steps) rather than the time spent per puzzle (we called running a test as making a puzzle, in the rest of this paper), then convert the step-cost into seconds according to the performance of the adopted computer. The tested platforms include: MacBook pro running OS X 10.12.5 equipped with 2 cores and 2.7GHz Inter Core i5 with 8GB DDR3 RAM. The experimental results are shown in the following:

Table 1. Average Crack times (in seconds) of the proposed SFEC-based time-lock encryption schemes with different orders.

bits of prime	32	36	40	44	48	52
Minimum	0.12	0.71	1.52	3.14	32.46	67.55
Average	3.08	10.62	47.21	146.36	684.45	2727.81
Maximum	8.56	39.64	161.42	506.27	2116.94	8918.28

As the experimental result shows, we can easily increase the average crack time of an SFEC-based time-lock encryption scheme by choosing elliptic curves with higher orders. But there is another problem: The difference between Max and Min crack times is very large; in other words, we may build a puzzle that could be compromised nearly instantly (say less than 0.2 seconds) or would take a long period of time (say more than 2 hours), because *Pollard's rho attack* has a certain degree of randomness in its behavior.

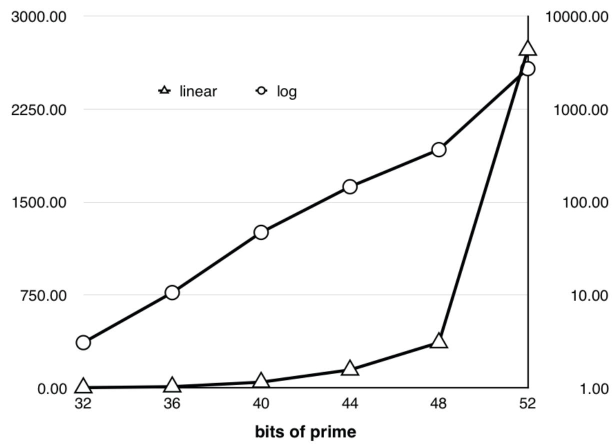

Figure 1. Plots of the average Crack times (in seconds), obtained from Table 1, in linear and logarithmic scales, respectively. The near constant slope of the log-scaled curve shows the average Crack time is direct proportional to the order of the chosen SFEC.

For solving the latest problem, we suggest converting a single puzzle with longer bit length to many puzzles with shorter lengths (it is called the **concatenation construction** from now on). For example, if we need to build a time-lock puzzle on a 52-bit elliptic curve, on the basis of a pre-defined security requirement (say a given elapsed time), then we would rather to concatenate four identical puzzles on a 48-bit elliptic curve, or sixteen identical puzzles on a 44-bit elliptic curve together, instead. More specifically, here we say concatenating puzzles together means putting one puzzle inside another.

Since the expected crack time of *Pollard's rho attack* is proportional to $\boldsymbol{O}(\sqrt{n})$, where n is the order of the chosen elliptic curve, the above puzzle constructions will produce the same crack-time complexity. In addition, creating multiple puzzles doesn't complicate the key generation procedures. What we need to do is just hashing the result of the previous shorter puzzle's key generation protocol, with the same $\boldsymbol{H_e}$, again. For checking the effectiveness of our proposed "putting puzzles inside another puzzle" mechanism, we re-calculate the Max, Average, and Min Key Cracking times, with different puzzle constructions. The results are shown in Table 2 and Figure 2.

Table 2. The Max, Average, and Min Key Cracking times (in seconds) with respect to different puzzle constructions, where "n x L" on the top of each column denotes "n puzzles, each with prime length L, are concatenated together". For showing the stability of our mechanism, the variances of the Cracking time are included also.

puzzle×difficulty	1×52	4×48	16×44	64×40	256×36	1024×32
Minimum	76	1103	1555	2316	2271	2432
Average	2587	2826	2407	3010	2569	2606
Maximum	8921	5354	3511	3774	2846	2786
Variance	2321879	661986	108383	57447	9289	3139

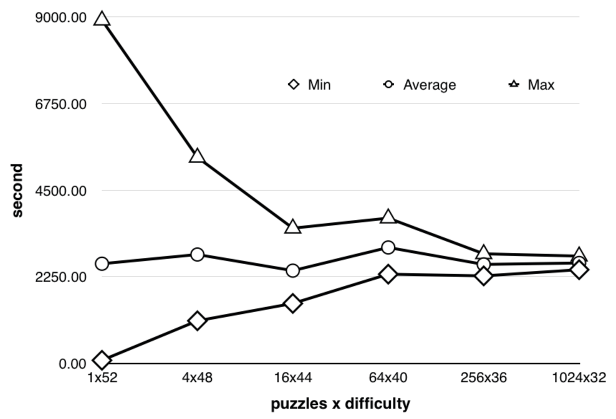

Figure 2. Plots of the Max, Average and Min Crack times for the puzzles with the same difficulty but different constructions. The much flat curve of the average crack-time shows the proposed "Concatenation Construction" mechanism effectively reduce the variation of crack time, caused by the randomness of *Pollard's rho attack.*

As shown in Figure 2, the more we turn a long-bit-length puzzle into concatenations of short-bit-length puzzles, the narrower the gap among the Max, Average, and Min Crack times is. This means, under the fixed computing power assumption, the concatenating construction will produce puzzles with much accurate expected compromised time. With the aid of this mechanism, participants without trust among each other, can build a public key system for encrypting sensitive message in a short period of time; moreover, they will have strong confidence that the encrypted messages will never be decrypted only when the scheduled release time is reached.

V. Integration with the Blockchain

The remaining challenging problems are "Who is willing to solve the puzzles for others?" and "How to set the appropriate difficulty of puzzles?", the second one is the same as "How to predict the total amount of required computing power?".

As reference [11] pointed out, with reasonable setting of incentives, one could utilize the blockchain technique coupling with a modified proof of work (PoW) scheme to solve the above-mentioned problems.

The PoW schemes adopted by Bitcoin and Ethereum [12] are similar to each other. That is, miners collect transactions made by other peers, in which a specific transaction is designated for paying a certain amount of rewards to the efficacious miner (i.e., the miner who mined the block). Then, a nonce is increased together with the state of the block to make their hash value less than a preset target (difficulty) value. Anyone, in the blockchain network, who finds the valid nonce will have the rights for appending the new block to the end of the main chain and getting the pre-described reward. Notice that even more than one of the miners may collect the same transaction, valid nonce for different miners are different because the addresses (included in the blocks' state information) of different miners will never be the same.

Now, if we naively change the mining scheme from "Finding a nonce to make the corresponding hash value less than the target value" to "Finding an appropriate private key of a given public key on SFEC", then the proposed fully decentralized time-lock encryption scheme could be combined with the blockchain to benefit from its well-known properties, such as Immutability, Transparency, and Searchability.

As mentioned in Section 3.2, the difficulty in reaching consensus on Bitcoin or Ethereum can be released via the generation of new blocks automatically, in our scheme, the difficulty of puzzle can be set in a similar way to stabilize the unlock time of sensitive messages. More specifically in our consensus-reaching scheme, lots of public points (coordinates) on a chosen SFEC are tested at once for supporting concatenation construction scheme. That is, anyone who finds the corresponding private key of a given public key will get the reward and has the rights to append the newly generated block to the end of the main chain. Of course, the difficulties of puzzles will be adjusted, empirically and periodically, according the exact time limits defined in practical applications, so as to decrypt the time-sensitive messages very much on schedule.

There is another interesting property of our scheme: miners who solving the puzzles have nothing to do with the sensitive information besides the mining rewards. That is, whatever the encrypted message is, there are always incentives for miners to contribute their computing resources. On the other hand, anyone who wants to publish his time-sensitive message (after the time limit) can just leave the encrypted message on the blockchain we designed. He needs not to worry about if his message will be decrypted or not. This is because miners will always find the private key for their own benefits.

There is still another problem needs to be dealt with. Since the private key coinciding with a given public key is unique, once a miner A who solved the puzzle is trying to broadcast the answer to other miners for getting his reward, any of the other miners, say B, can fake that he is the first one who solved the puzzle by rebroadcasting the answer. Then, the whole network would not be synchronized since miners are selfish in nature. For dealing with this problem, we assume there is an encrypted message $\boldsymbol{E_k(m)}$, where $\boldsymbol{k}$ is the public key used to encrypt the time-sensitive message $\boldsymbol{m}$. Then for a miner who found $\boldsymbol{x}$ such that $\boldsymbol{xG = k}$, where $\boldsymbol{G}$ is a base point on the chosen SFEC, will broadcast the decrypted message $\boldsymbol{m}$ combined with a zero-knowledge proof [13] such that no one can fake the answer he found.

In our concatenating construction scheme, a participant who wants to set a long period time capsule has to design a lot of puzzles. As a consequence, we have to work out many puzzles at once. For instance, if the average time of appending a new block to the main chain is assumed to be 1 hour and requires 10 puzzles. For the user who wants to open his message after 24 hours has to find 240 puzzles for encrypting his message.

However, to release such many puzzles at the same time might let some miners be able to mine the blocks which should be mined later in the chain (i.e., miners might solve puzzles with timestamps, or orders, later than that of the current puzzle). Fortunately, depending on the blocks' generation time, one can still adjust the preset difficulties of the to-be-solve puzzles, such that both the objectives of secret block mining and speeding up the block generation can be achieved simultaneously. In other words, even though many puzzles are released, at the same time, and some dishonest miners did scoop the ahead-of-schedule blocks out, we still have strong confidence that the locked messages won't be decrypted earlier than the pre-defined schedule.

Notice that those miners who mined the ahead-of-schedule blocks won't get extra rewards because in our concatenating constructing scheme, the rewards of miners are linearly proportional to the computing power they offered rather than their luck, because the gap between the Maximal and the Minimal solving times has been significantly narrowed and is closely approaching to the average solving time.

By providing the pre-described steady key generation mechanism, our system can successfully embed a time-lock encryption scheme into a blockchain network, equipped with the modified PoW algorithm, allowing users to set an encrypted message with appropriate difficulty and miners to solving puzzles by users' voluntariness. This decentralized time-lock encryption system has the following preferred features:

1. Providing reliable digital currency flow,
2. Building trust without relying trusted third parties,
3. Supporting decentralized time lock function,
4. Stabilizing the variation of decryption time,
5. Being integrable with smart contracts.

Clearly, the proposed system founds a good timing basis for building a worldwide on-line auction and/or E-voting platforms.

VI. Conclusion

With the proposed time-lock encryption scheme, mutually untrusted users are able to generate a common public key for conducting various time critical privacy preserving applications, such as E-voting or on-line auction, in a fully distributed and asynchronous environment, like the Internet or blockchain networks. To deal with the probabilistic characteristics of the expected key cracking time, a "putting puzzles inside another puzzle" construction mechanism is proposed. Experimental results show that the proposed method can produce a more accurate key cracking time, which helps us to meet the scheduled deadline more easily and accurately.

As mentioned in Section V, we can further estimate computational power for decrypting time-sensitive message via combining our scheme with the blockchain. On the basis of the modified PoW scheme, miners are willing to provide computing power for others, make the blockchain more suitable for building a decentralized worldwide time-lock system.

References

[1] Tripathi, Ritu, and Sanjay Agrawal. "Comparative study of symmetric and asymmetric cryptography techniques." International Journal of Advance Foundation and Research in Computer (IJAFRC) 1.6 (2014): 68-76.

[2] Rivest, Ronald L., Adi Shamir, and David A. Wagner. "Time-lock puzzles and timed-release crypto." (1996).

[3] Mahmoody, Mohammad, Tal Moran, and Salil Vadhan. "Time-lock puzzles in the random oracle model." Annual Cryptology Conference. Springer, Berlin, Heidelberg, 2011.

[4] Cathalo, Julien, Benoît Libert, and Jean-Jacques Quisquater. "Efficient and non-interactive timed-release encryption." International Conference on Information and Communications Security. Springer, Berlin, Heidelberg, 2005.

[5] Rabin, Michael O., and Christopher Thorpe. "Time-lapse cryptography." (2006)..

[6] Jager, Tibor. "How to Build Time-Lock Encryption." IACR Cryptology ePrint Archive 2015 (2015): 478.

[7] Steiner, Michael, Gene Tsudik, and Michael Waidner. "Diffie-Hellman key distribution extended to group communication." Proceedings of the 3rd ACM conference on Computer and communications security. ACM, 1996.

[8] Morain, François. "Building cyclic elliptic curves modulo large primes." Workshop on the Theory and Application of of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1991.

[9] Icart, Thomas. "How to hash into elliptic curves." Advances in Cryptology-CRYPTO 2009. Springer, Berlin, Heidelberg, 2009. 303-316.

[10] Hankerson, Darrel, Alfred J. Menezes, and Scott Vanstone. Guide to elliptic curve cryptography. Springer Science & Business Media, 2006.

[11] IME-LOCK ENCRYPTION, https://www.gwern.net/Self-decrypting-files

[12] Ethash, https://github.com/ethereum/wiki/wiki/Ethash

[13] Rackoff, Charles, and Daniel R. Simon. "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack." Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 1991