

*Mini Project Report on*

# **A FULLY DECENTRALIZED TIME-LOCK ENCRYPTION SYSTEM ON BLOCKCHAIN**

for the course

**IT465 : Cryptocurrencies and Blockchain Technologies**

*Submitted By*

**Ishan Mathew Nedumkunnel (171IT217)**

**Linu Elizabeth George (171IT220)**

**VIII SEM B.Tech (IT)**

Under the Guidance of,

**Dr. Bhawana Rudra**

**Dept of IT, NITK Surathkal**

in partial fulfillment for the award of the degree

of

**Bachelor of Technology**

In

**Information Technology**

At



**Department of Information Technology**

**National Institute of Technology Karnataka, Surathkal**

**November 2020**

## Abstract

Time-lock encryption is a method to encrypt a message such that it can only be decrypted after a certain deadline has passed. While there exist lots of studies on time-lock encryption, they may lack in the decentralised aspect of the lock. In this project, we aim to implement a decentralized time-lock encryption system on blockchain such that it is not dependent on any third parties. There are several applications of timed-release crypto that would be helped by this work.

**Keywords:** *Time-lock, Encryption, decentralised*

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Literature Survey</b>	<b>2</b>
2.1	Related Work . . . . .	2
2.2	Outcome of Literature Review . . . . .	4
<b>3</b>	<b>Requirements Analysis</b>	<b>5</b>
3.1	Functional Requirements . . . . .	5
3.2	Non Functional Requirements . . . . .	5
<b>4</b>	<b>System Architecture</b>	<b>6</b>
<b>5</b>	<b>Methodology</b>	<b>7</b>
5.1	Cryptography Aspect . . . . .	7
5.2	Blockchain Aspect . . . . .	8
<b>6</b>	<b>Implementation</b>	<b>9</b>
6.1	Technology Stack . . . . .	9
6.2	Solidity Smart Contract . . . . .	9
6.3	Snapshots of Implementation . . . . .	10
<b>7</b>	<b>Results and Analysis</b>	<b>15</b>
<b>8</b>	<b>Conclusion</b>	<b>16</b>

## List of Tables

1	Advantages and Disadvantages of different approaches . . . . .	4
---	--	---

## List of Figures

2.1	Key Exchange Diagram . . . . .	3
4.1	System Architecture . . . . .	6
6.1	Application Layout after smart contract has been deployed via truffle . .	10
6.2	Users enter random seed value to Generate Public Key . . . . .	10
6.3	Approve this transaction via Metamask . . . . .	11
6.4	The Public Key is generated after passing the seed through a hash function	11
6.5	Pollard Rho attack using given prime and primitive root . . . . .	11
6.6	Square and Multiply to verify locally that it is the correct private key. We can see that the output is equal to the public key. . . . .	12
6.7	Entering a wrong private key . . . . .	12
6.8	Smart Contract checks and does not publish the wrong private key; miner is not rewarded . . . . .	12
6.9	The correct private key is entered . . . . .	13
6.10	Smart Contract verifies the private key and publishes it onto the blockchain	13
6.11	Events on Ganache . . . . .	14
6.12	Published Private Key . . . . .	14

# 1 Introduction

In many scenarios, we may find ourselves wanting to keep information secure up until a particular time, after which we wish it to be easily accessible. Time-lock encryption achieves the following properties simultaneously:

1. **Decryption is non-interactive**
2. **No trusted setup**
3. **No resource restrictions**

The applications of timed release crypto are as follows:

- A bidder in an auction will want to seal his bid, so that it may only be opened after the bidding period is closed.
- In the case of a homeowner, they may want to give their mortgage holder a series of encrypted mortgage payments. These might be encrypted digital cash with different decryption dates so that one payment becomes decryptable and thus usable by the bank at the beginning of each successive month.
- A person may want to secure and encrypt his diaries so that they may only be decrypted after a particular number of years.
- A key-escrow scheme may be based on time-lock encryption so that the government can get the message keys but only after a fixed period.

## 2 Literature Survey

### 2.1 Related Work

The problem of a time-lock encryption was first discussed by Timothy May [1]. The intuitive solution to a time-lock encryption is to rely on a trusted third party. This party reveals the decryption keys at a pre-decided time. A study by Rabin et al [2] proposes this very solution.

The term 'Time-lock puzzles' are used to refer to computational puzzles that require a precise amount of time (real time, not CPU time) to solve. The solution to the puzzle gives a key which can be used to decrypt the secured information. A study conducted by Rivest et al [3] suggested a way to create time-lock puzzles and use them to time-lock encrypted data. A major difficulty in such an approach is that those with more computational resources might be able to solve the timelock puzzle more quickly by using large parallel computers. In their study, they have tried to overcome this by making their time-lock puzzles "intrinsically sequential". However, this method puts a considerable computational overhead on the receiver.

Liu et al [4] 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 cyberspace can encrypt his messages with the time specific key he generated. Anyone who wants to decrypt the messages has 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.

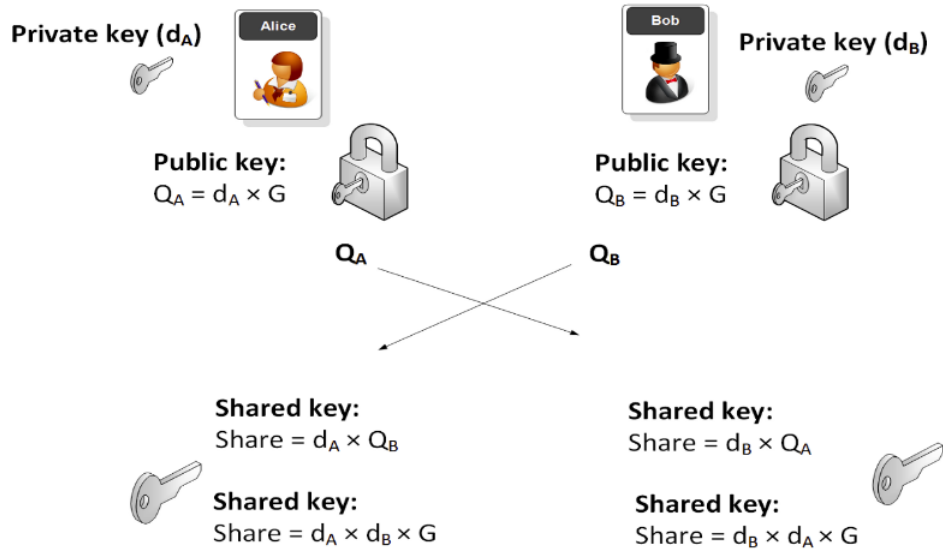


Figure 2.1: Key Exchange Diagram

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  $xG$ , Bob generates  $yG$ , and both of them open  $xyG$  to the public, where  $G$  is a base point of the elliptic curve. Now, participants can encrypt their sensitive messages with this public key  $xyG$ . Steiner et al proposed [5] a method that involved increasing the number of partial secret holders from two to many to make the process more distributed. This approach allows us to build a public key for encrypting secrets without considering the corresponding private key

## 2.2 Outcome of Literature Review

Table 1: Advantages and Disadvantages of different approaches

Paper	Advantages	Disadvantages
Rabin et al [2]	Straightforward solution	Third party may not be trustworthy
Rivest et al [3]	No need of a trusted third party	Puts a considerable computational overhead on the receiver
Steiner et al [5]	No need of a trusted third party	-Once all of the partial key holders collude together, the sensitive messages will be publicly aware before the scheduled time. -Once one of the partial key holders does not open or loses his secret, the encrypted messages will never be opened again.



## 3 Requirements Analysis

### 3.1 Functional Requirements

1. **Decryption is non-interactive:** The sender is not required to be available for decryption.
2. **No trusted setup:** Time lock encryption does not rely on trusted third parties. Thus, the sender is not required to trust any (set of) third parties to keep decryption keys (or shares of decryption keys) secret until the deadline has passed.
3. **No resource restrictions:** Parties interested in decrypting a ciphertext are not forced to perform expensive computations until decryption succeeds. This means that a party which simply waits till the decryption deadline has passed will be able to decrypt the ciphertext at about the same time as another party who attempts to decrypt the ciphertext earlier by performing a large (but reasonably bounded) number of computations. Thus, all reasonably bounded parties will be able to decrypt a ciphertext at essentially the same time, regardless of their computational resources.

### 3.2 Non Functional Requirements

1. **Scalability:** Blockchain scalability is tied up to its underlying platform.
2. **Simplicity:** The UI must be simple and clear even to someone who has no clear knowledge of what goes on behind the scenes.

## 4 System Architecture

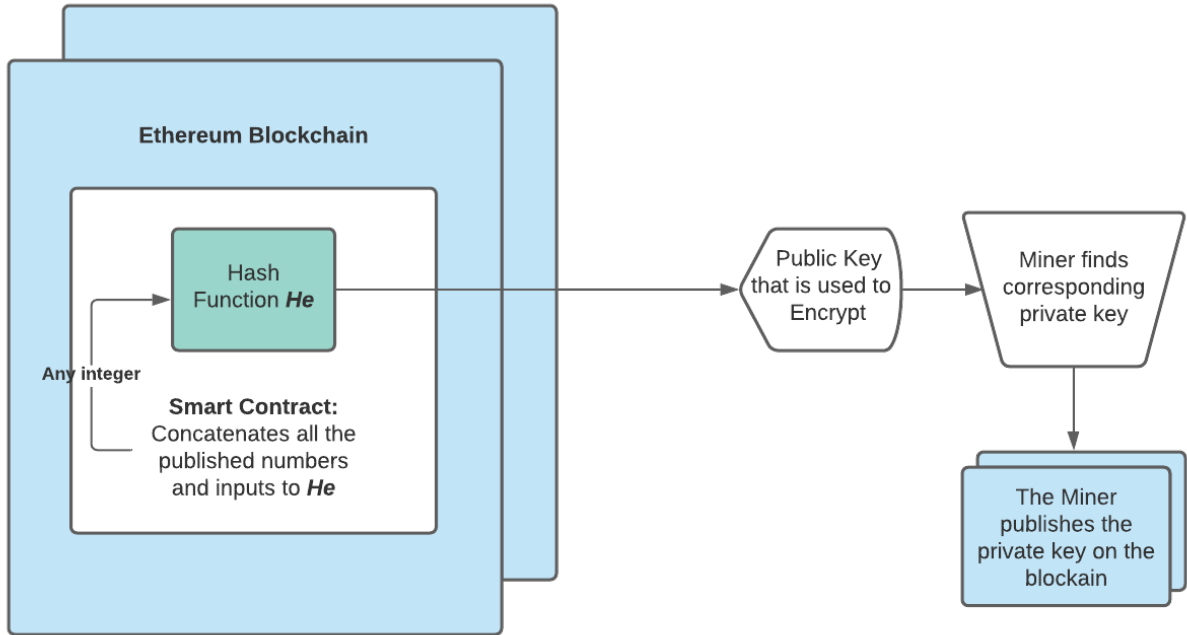


Figure 4.1: System Architecture

The Hash Function output is the public key is used by everyone to encrypt. The Miner solves a semi-feasible Discrete Logarithmic Problem on a set of parameters determined by the owner of the contract, perhaps using the Pollard-Rho attack, which takes  $O(\sqrt{N})$  time. The set of parameters chosen by the owner vary according to the time for which the message(s) need to be locked. This solution, which acts as the replacement for the POW algorithm, is published on the blockchain. Everyone can use this published private key to decrypt the secured information.

## 5 Methodology

### 5.1 Cryptography Aspect

We need to generate a key pair that can be used for encryption/decryption. In this case, public key is generated without the use of a private key, unlike standard key exchange algorithms.

The discrete logarithm problem (DLP) is the following computational problem: Given integers  $0 \leq p, q < n$ , where  $n$  is a prime and  $p$  is a primitive root of  $n$ ; to find an integer  $a$ , if it exists, such that  $q = ap \pmod{n}$ .

The idea is to use a semi-feasible DLP. Using the Pollard-Rho attack, the DLP can be solved in  $O(\sqrt{N})$  time.  $N$  can be chosen in such a way that this time complexity meets the desired time after which the secret can be exposed.

A hash function  $H$  is used, that maps a string of user-given seed values to a number lying between 0 and the chosen prime. This point is then treated as Public Key, and the search for a private key to match this public key is started.

$H: x \rightarrow y \pmod{n}$  where  $x$  is the seed and  $y \pmod{n}$  is the public key.

The public Key is made available to everyone, they can use this to encrypt their messages. Decryption is not possible until the corresponding private key is found. The time taken to find this private key can be designed by increasing the value of the prime number  $n$ .

## 5.2 Blockchain Aspect

After the public key is released, miners try to search for the private key - this is similar to solving a proof of work puzzle. The miner that finds the private key first is rewarded, and the private key is also published on the blockchain. After everyone has the private key, the sensitive information is now disclosed to everyone - which is what we desire.

The overall steps are as follows:

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.
2. An Ethereum-based smart contract concatenates all the numbers published in step 1 together, as an input to  $\mathbf{H}_e$ , and obtains a point on SFEC.
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.

## 6 Implementation

### 6.1 Technology Stack

The tech stack that we used to implement this project was

- Ethereum - Our blockchain was powered by Ethereum.
- Truffle - We used Truffle as our development framework to create the Ethereum Blockchain.
- Ganache - A personal blockchain for Ethereum development that we used to deploy contracts, develop our application, and run tests.
- Solidity - Our smart contract was coded in Solidity.
- HTML - For front-end
- Javascript - For front-end

### 6.2 Solidity Smart Contract

The smart contract written in solidity is the crux of our blockchain. We have considered that the deployer of the smart contract is the 'owner' and thus gets to decide how long he wants his time-lock to be. The parameters that decide this, the prime and the primitive root, are thus built into the contract and cannot be changed once deployed. The contract takes in 'seeds' from multiple users for a specified duration written into the contract, which it uses to create the public key. Once this duration passes, the public key is fixed, and miners can solve the Discrete Logarithmic problem to find the private key.

The calculated private key is entered and this value is checked by the smart contract. Once the smart contract verifies, the private key is published on the blockchain and the miner is rewarded. If the wrong private key is entered, there will be no change of state.

## 6.3 Snapshots of Implementation

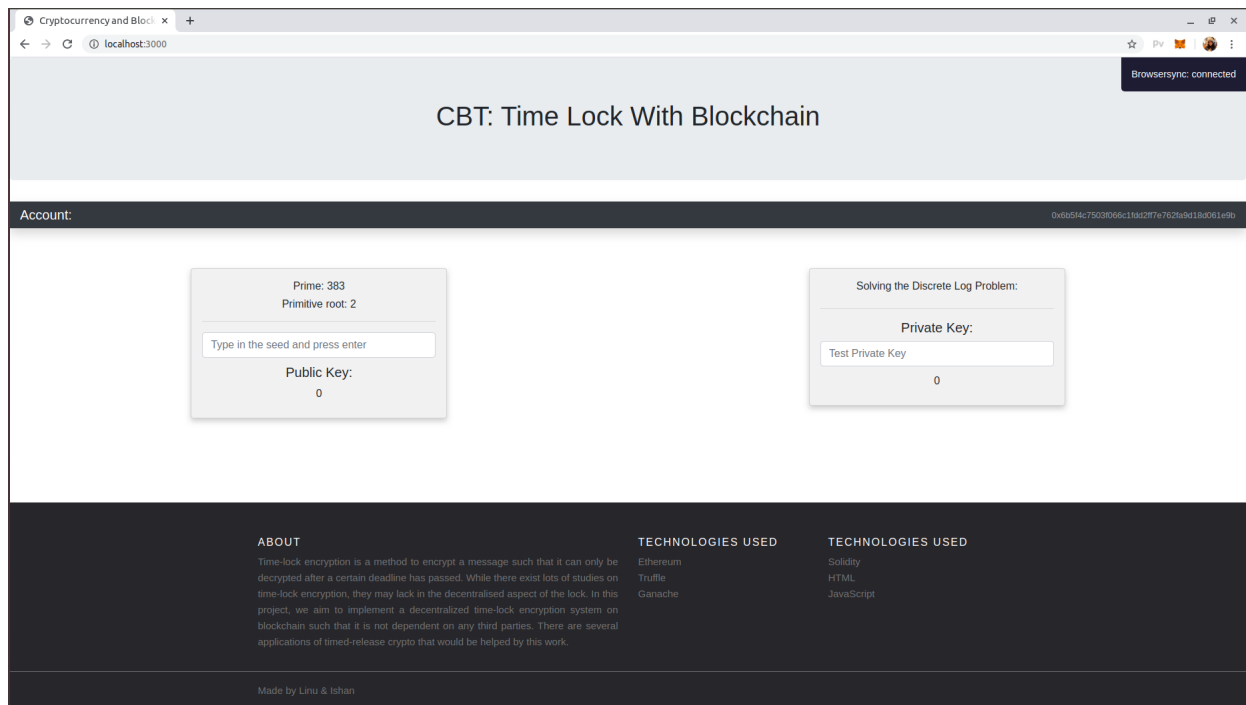


Figure 6.1: Application Layout after smart contract has been deployed via truffle

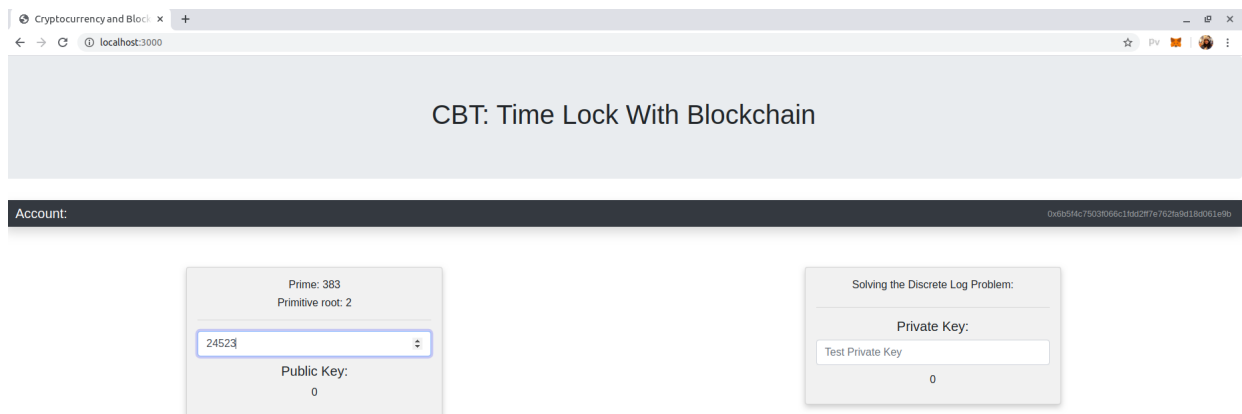


Figure 6.2: Users enter random seed value to Generate Public Key

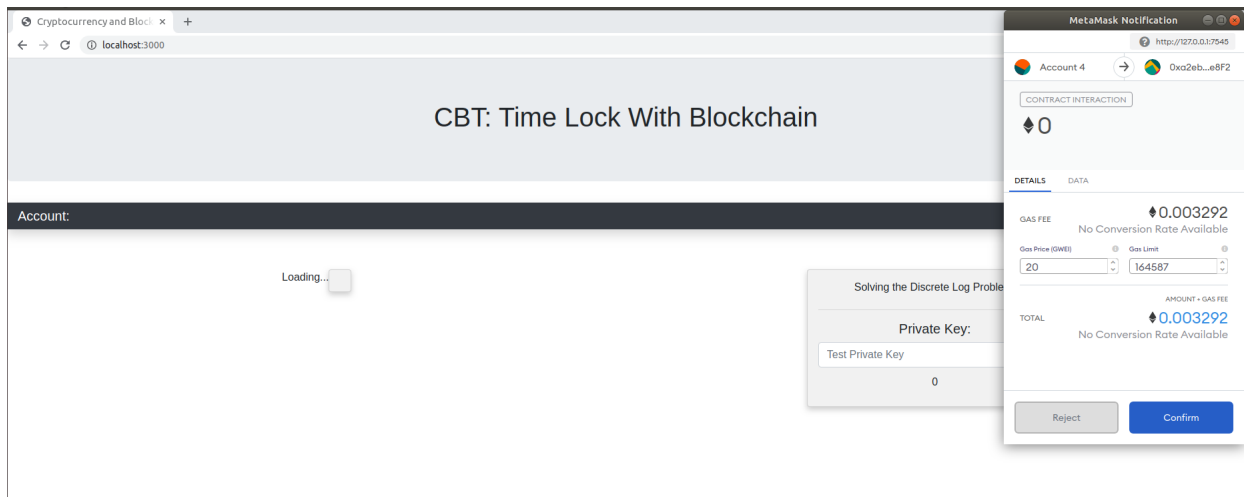


Figure 6.3: Approve this transaction via Metamask

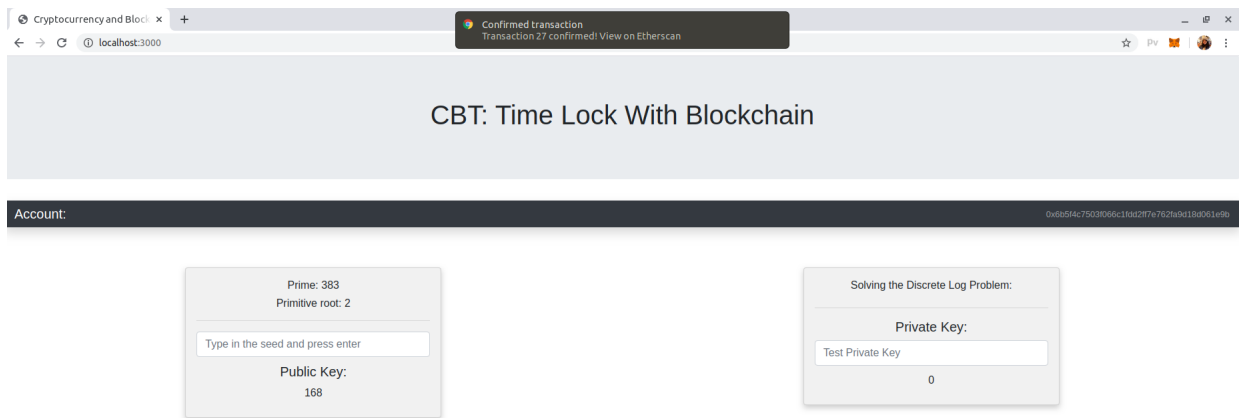


Figure 6.4: The Public Key is generated after passing the seed through a hash function

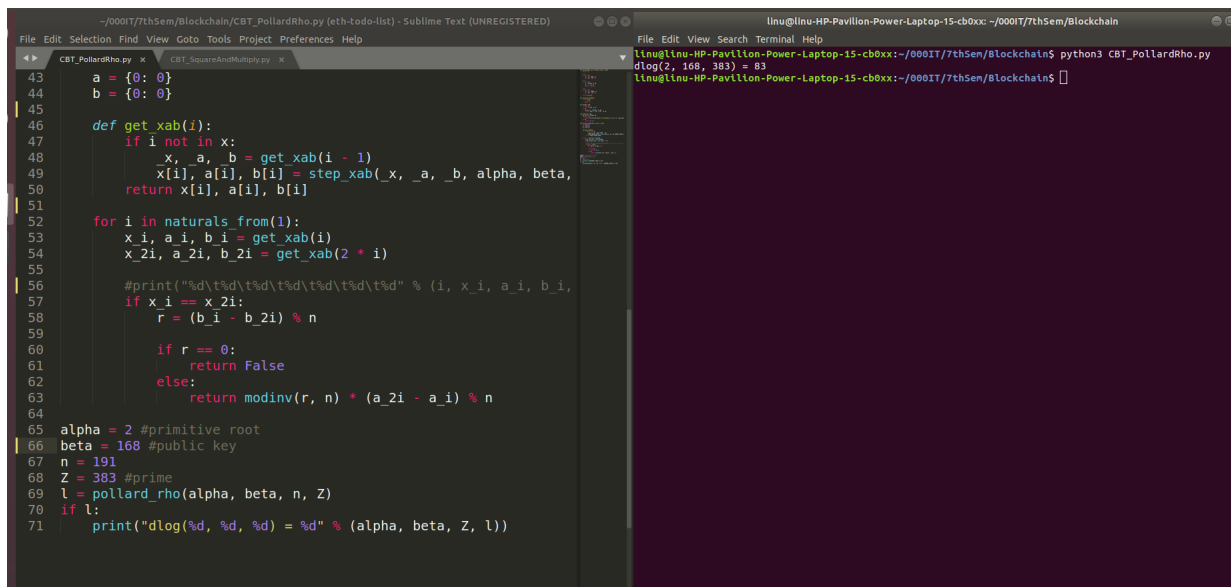
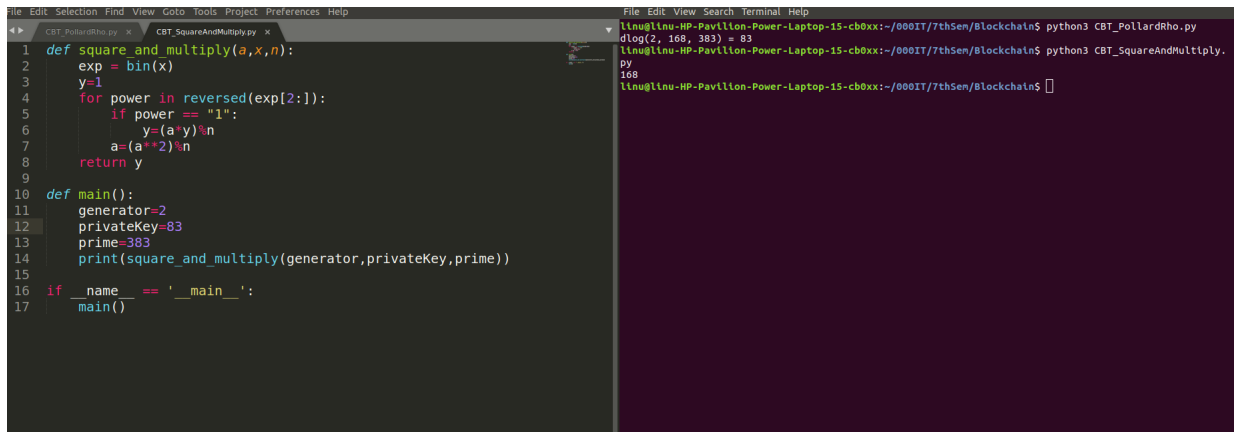


Figure 6.5: Pollard Rho attack using given prime and primitive root



The image shows a code editor with two files: `CBT_PollardRho.py` and `CBT_SquareAndMultiply.py`. The `CBT_PollardRho.py` file contains the following code:

```
1 def square_and_multiply(a,x,n):
2     exp = bin(x)
3     y=1
4     for power in reversed(exp[2:]):
5         if power == "1":
6             y=(a*y)%n
7             a=(a**2)%n
8     return y
9
10 def main():
11     generator=2
12     privateKey=83
13     prime=383
14     print(square_and_multiply(generator,privateKey,prime))
15
16 if __name__ == '__main__':
17     main()
```

The terminal shows the execution of the code:

```
linu@linu-HP-Pavilion-Power-Laptop-15-cb0xx:~/000IT/7thSen/Blockchain$ python3 CBT_PollardRho.py
dlog(2, 168, 383) = 83
linu@linu-HP-Pavilion-Power-Laptop-15-cb0xx:~/000IT/7thSen/Blockchain$ python3 CBT_SquareAndMultiply.py
168
linu@linu-HP-Pavilion-Power-Laptop-15-cb0xx:~/000IT/7thSen/Blockchain$
```

Figure 6.6: Square and Multiply to verify locally that it is the correct private key. We can see that the output is equal to the public key.

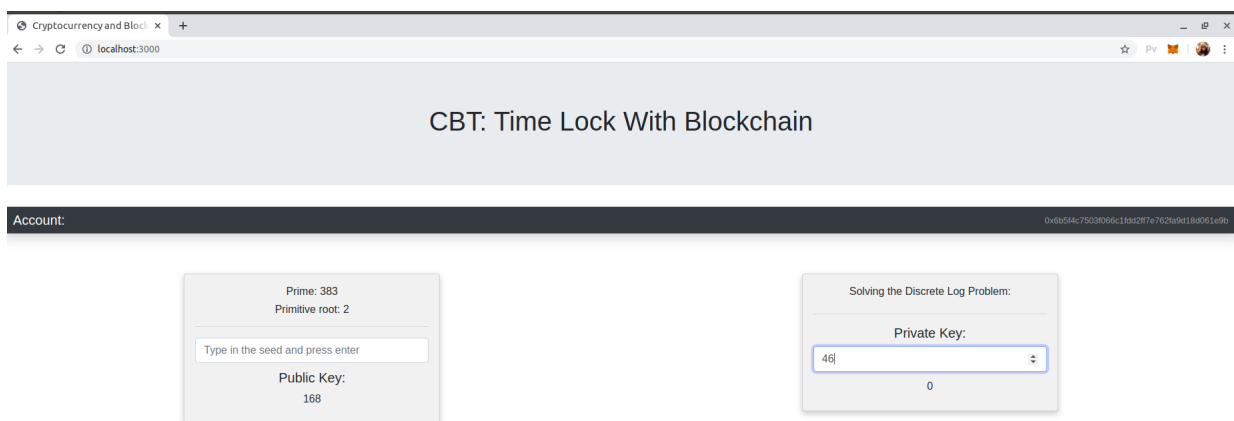


Figure 6.7: Entering a wrong private key

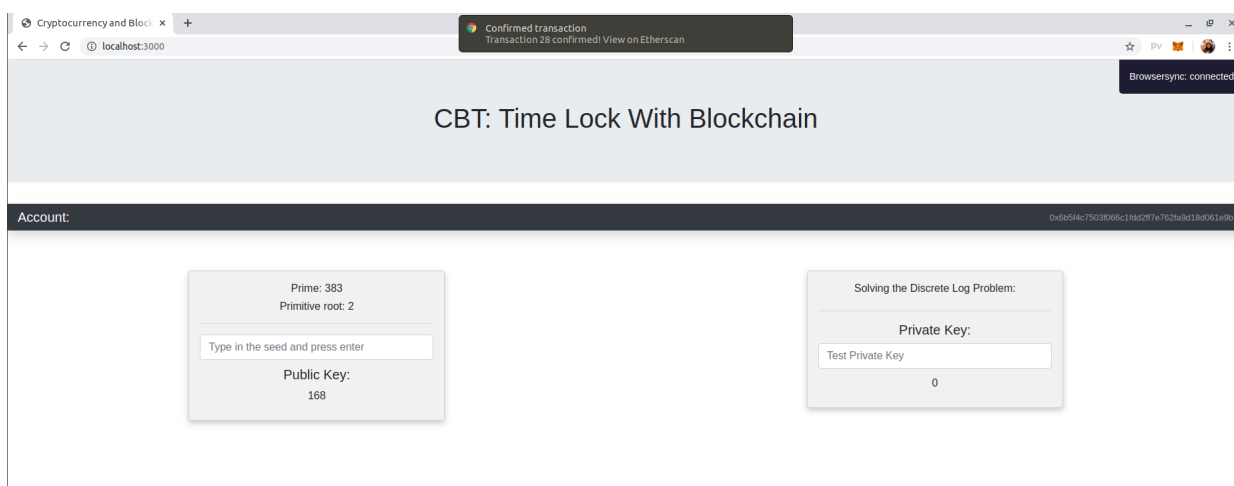




Figure 6.8: Smart Contract checks and does not publish the wrong private key; miner is not rewarded

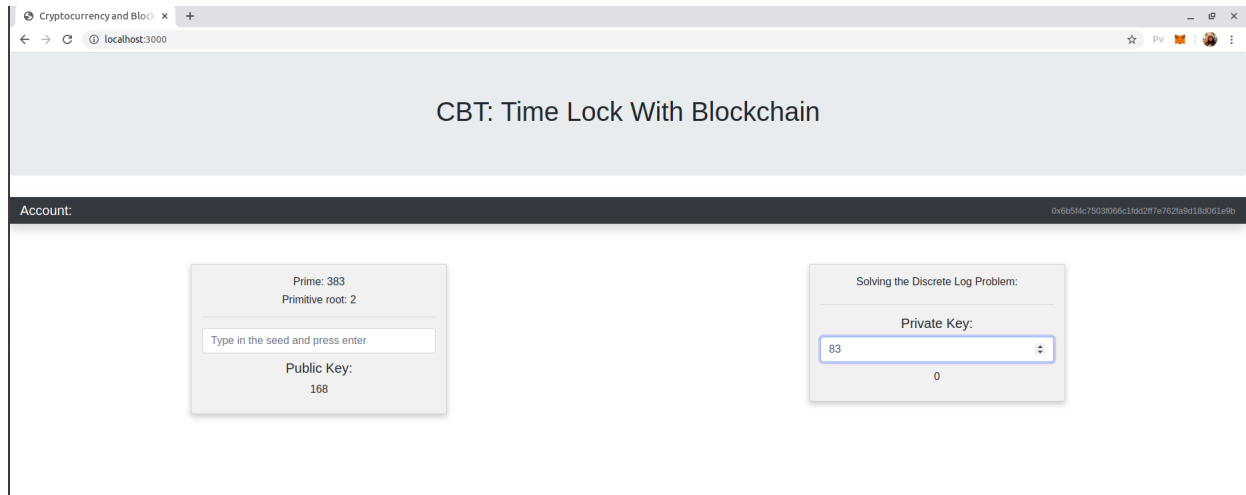


Figure 6.9: The correct private key is entered

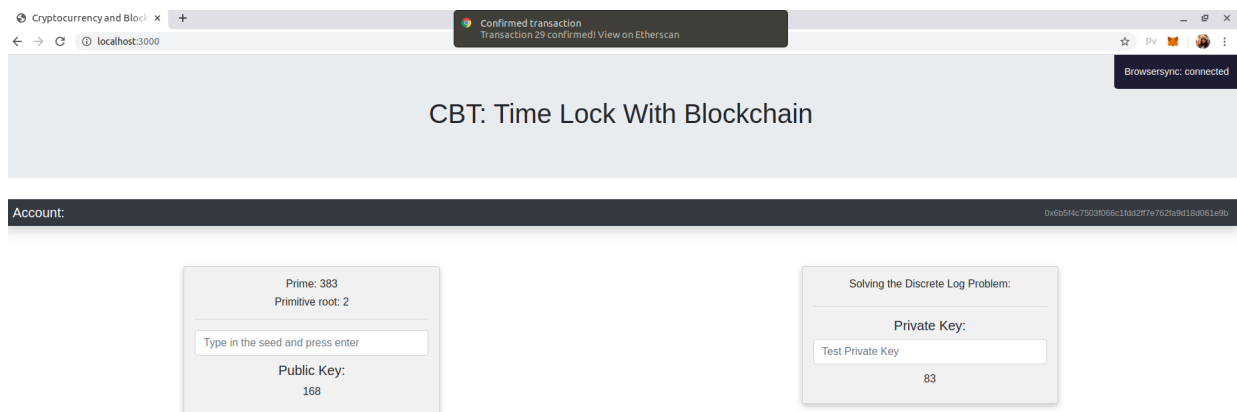


Figure 6.10: Smart Contract verifies the private key and publishes it onto the blockchain

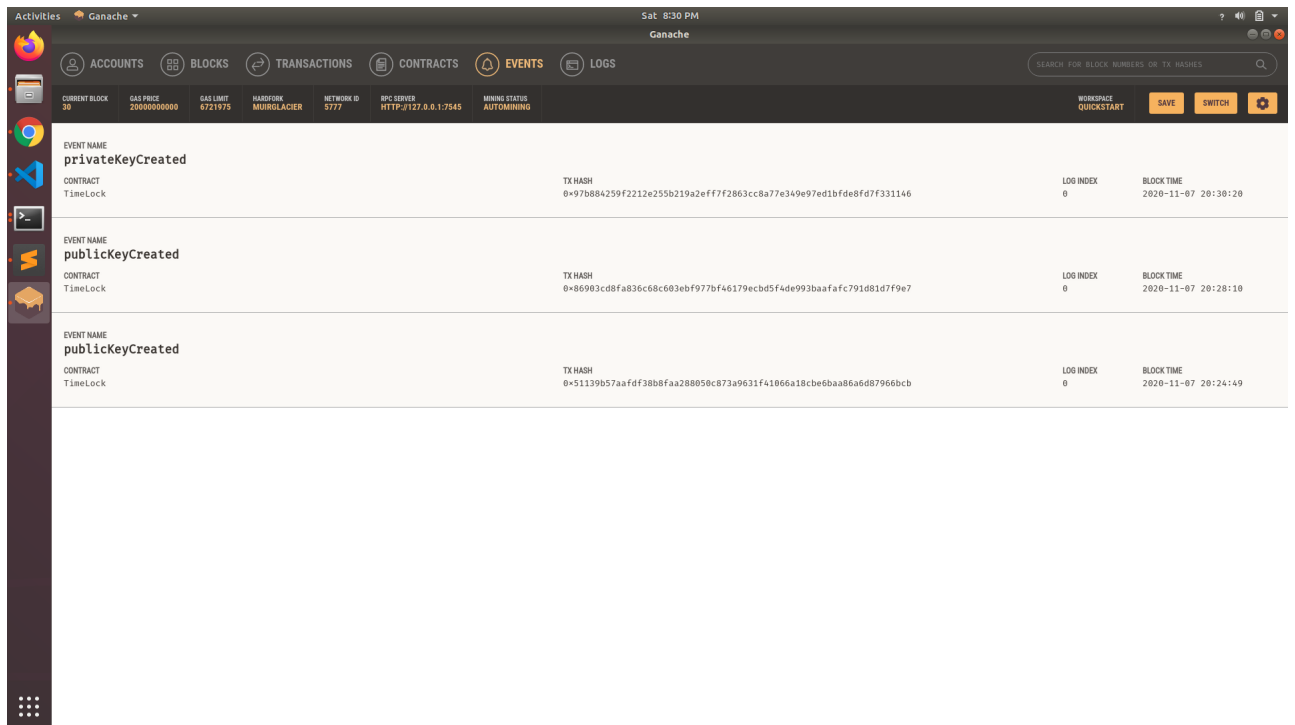


Figure 6.11: Events on Ganache

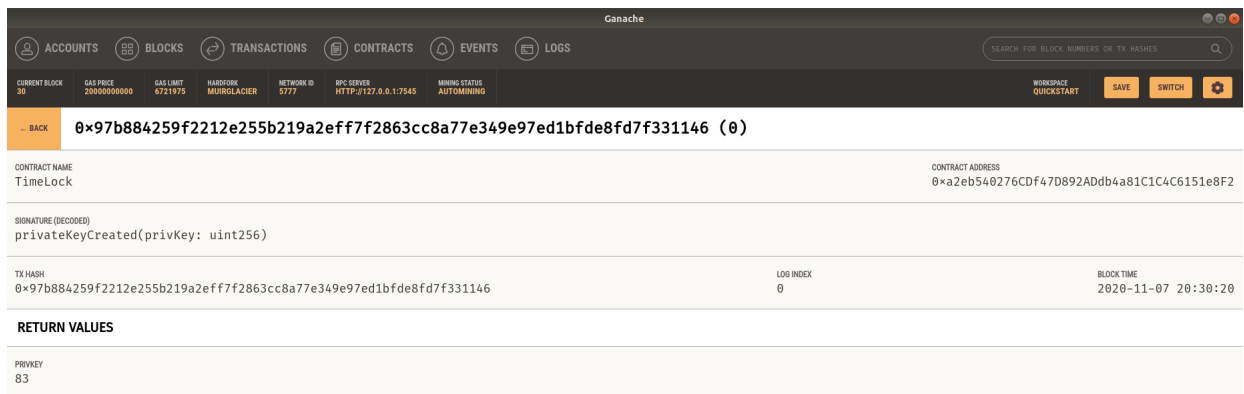


Figure 6.12: Published Private Key

## 7 Results and Analysis

We have implemented the paper and our application and its working can be seen through the screenshots. The deployer of the contract can adjust the values of the prime and the primitive root to adjust the time for which he wants his information to be encrypted. We have also included the facility for multiple users to input their 'seed' values and calculate the public key based on these values. Once the private key is calculated, verified and published, anyone who used the public key to encrypt can use this private key to decrypt.

The novelty of our approach is the use of the general and more simple version of the Discrete Logarithm Problem, without the use of an Elliptic Curve. The complexities that come with the usage of the Elliptic Curve do not arise in our approach, and are anyway not required to be dealt with for our use-case. The bounding time-complexity of the Pollard-Rho attack can also be calculated more accurately if this simpler version of the DLP is used.

## 8 Conclusion

The proposed method involving a semi-feasible elliptic curve and integration with blockchain satisfies all the requirements of time-lock encryption. The sender's presence is not required, there is no third party, and as the private key, once solved by the miner is published on the Blockchain, the data can be encrypted by anyone regardless of their computational resources. Blockchain provides an apt solution for a secure, yet decentralised system.

**This proposed system, as mentioned previously, will have numerous applications in cases where one deals with time-sensitive data.**

## References

- [1] Timothy C May Timedrelease crypto February  
<http://cypherpunks.venona.com/date/1993/02/msg00129.html>
- [2] Rabin, Michael O., and Christopher Thorpe. "Time-lapse cryptography." (2006).
- [3] Rivest, Ronald Shamir, Adi Wagner, David. (2001). Time-lock puzzles and timed-release Crypto.
- [4] Liu, Jia Jager, Tibor Kakvi, Saqib Warinschi, Bogdan. (2018). How to build time-lock encryption. Designs, Codes and Cryptography. 10.1007/s10623-018-0461-x.
- [5] 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.