Loamit: A Blockchain-based Residual Loanable-limit Query System

Lijing Zhou[a], Licheng Wang[a,*], Yiru Sun[a], Pin Lv[a]

[a]*the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Bei Jing* 100876*, P.R. China.*

Abstract

Currently, the blockchain technology is experiencing an exponential growth in the academia and industry. Blockchain may provide the fault-tolerance, tamper-resistance, credibility and privacy to users. In this paper, we propose a blockchain-based residual loanable-limit query system, called Loamit. Firstly, to the best of our knowledge, it is the first work to prevent that a client, who lacks the ability to repay, borrows an un-repayable amount of money from independent banks without releasing the personal privacy of client. Specifically, if a client wants to borrow a certain amount of money from a bank, then the bank can get the client's residual loanable-limit in the alliance of banks without knowing details of the client's previous loans and repayments. Secondly, most of data in Loamit is verifiable. Therefore, malicious banks can be checked out. Thirdly, Loamit is fault-tolerant since it may work smoothly as long as a certain number of banks are active and honest. Finally, we deploy the Loamit system on the Ethererum private blockchain and give the corresponding performance evaluation.

Keywords: Blockchain, loanable-limit, bank, verifiable secret sharing.

*Corresponding author
Email address: wanglc2012@126.com (Licheng Wang)

1. Introduction

In daily life, there could exist a case that a client borrows a very huge amount of money from many different banks, but the client does not have the ability to repay so much money. It causes a certain degree of loss to banks. This case could appear if the banks work independently. In fact, this problem can be easily resolved by allowing that every bank can honestly tell other banks how much money the client has borrowed from the bank. However, we know that it is impractical since this method will face several challenges as follows:

- If some banks are malicious, off-line or damaged, then the method might be noneffective.
- A malicious bank might modify or delete a client's records of loans and repayments.
- A bank cannot judge whether the data sent by other banks has been correctly computed by these banks.
- It releases the personal privacy of clients and business privacy of banks.

Blockchain, which is first proposed by Bitcoin [1], is a tamper-resistant timestamp ledger of blocks that is utilized to share and store data in a distributed manner. Blockchain has attracted enormous attention from academics and practitioners (e.g., computer science, finance and law) [2]. The advantages of blockchain for addressing the above challenges are as follows:

- **Fault-tolerance.** Blockchain network can work smoothly, although a certain number of record-nodes might be malicious, off-line or damaged.
- **Tamper-resistance.** Once some data has been recorded in the blockchain (based on PBFT consensus [3]), it cannot be modified or deleted.
- **Credibility.** If some data has been recorded in the blockchain, then the publicly verifiable part of the data is credible.

- **Privacy.** Identities and transferred messages can be hidden by using various cipher technologies such as zk-SNARKs (used in Zcash [4]), RingCT (used in Monero [5]) and Bulletproof [6].

Therefore, a blockchain-based residual loanable-limit query system may be a reasonable choice to address the problem of fraudulent loan. In this system, all banks should form an alliance. Moreover, the *residual loanable-limit* indicates the residual most-amount of that a client can borrow from the alliance after loans and repayments. Besides, the system should additionally satisfy the following features:

- If a client's residual loanable-limit is a positive number, then the whole alliance can lend the client at most the positive number of money. In contrast, if a client's residual loanable-limit is zero or negative, then the whole alliance will not lend the client.

- A loan/repay business will confidentially reduce/increase the residual loanable-limit. In this process, except the related bank, other banks cannot learn what has happened.

- A bank can get a client's residual loanable-limit by inquiring other banks. However, in this process, the bank cannot get any information except the residual loanable-limit.

If the method is performed, then it may significantly prevent that a client, who lacks the ability to repay, borrows an un-repayable amount of money from independent banks as well as protects the personal privacy of client and business privacy of bank.

Our contributions. In summary, the contributions of this paper are as follows:

1. We propose a blockchain-based residual loanable-limit query system, called Loamit. In the Loamit system, amounts of client's loans and repayments are confidentially shared among an alliance of banks. Moreover, a bank

of the alliance may confidentially obtain a client's residual loanable-limit by inquiring other banks. Particularly, in this process, the bank cannot obtain anything except the residual loanable-limit. After that, according to the client's residual loanable-limit, the bank can identify whether it can to lend this client and how much money it can lend this client. To the best of our knowledge, it is the first work to prevent that a client, who lacks the ability to repay, borrows an un-repayable amount of money from independent banks without releasing the personal privacy of clients and the business privacy of banks.

2. In Loamit, we use the verifiable secret sharing to share the amount of loan or repayment among banks. Therefore, malicious banks can be checked out. Moreover, due to the threshold of secret sharing, the Loamit system can work smoothly, although a certain number of banks are malicious, off-line or damaged.

3. A prototype system is implemented to evaluate the feasibility of Loamit. Specifically, the prototype system is built upon the Ethereum private blockchain with four record-nodes. Moreover, we use the transaction simulator to simulate banks to generate and send transactions to evaluate the performance of Loamit.

Organization. The remainder of the paper is organized as follows. An overview of Loamit is shown in Sec.II. Sec.III briefly introduces preliminaries. The system setting and model is discussed in Sec.IV. We describe the construction of the Loamit system in Sec.V. Scenario is set up for performance evaluation in Sec.VI. Finally, a short conclusion is given in Sect.VII.

2. An Overview of Loamit

Loamit is a blockchain-based residual loanable-limit query system, where all related data is recorded in the blockchain and all banks only trust data recorded in the blockchain. Loamit is composed of an alliance of banks. For a new client, the bank, who deals with the first bank business for the client

in the bank alliance, will set an initial loanable-upper-limit $Limit_{upper}$ to the client. Assume that the client later borrows k times and repay t times in some banks of the alliance. Let the borrow-amounts be $M_{b,1}$, $M_{b,2}$ ... $M_{b,k}$ and the repay-amounts be $M_{r,1}$, $M_{r,2}$ ... $M_{r,t}$. Consequently, the client's residual loanable-limit $Limit_{residual}$ can be computed as the following equation:

$$Limit_{residual} = Limit_{upper} + \sum_{i=1}^{k} M_{b,i} - \sum_{i=1}^{t} M_{r,i}.$$

To hide borrow-amounts and repay-amounts, Loamit works as follows. Essentially, the loanable-upper-limit, borrow-amounts and repay-amounts are secretly shared among the alliance by using the verifiable secret sharing. Specifically, for a new client, the bank, who deals with the first bank business for the client in the bank alliance, will set an initial loanable-upper-limit (positive number) to the client by sending a limit-transaction to the blockchain. The limit-transaction contains encrypted shares about the loanable-upper-limit. After that, other banks can confidentially get their shares from the limit-transaction, respectively. (In fact, the bank sending the limit-transaction can set different loanable-upper-limit to different clients according to the working background of the clients.)

If the client later borrows coins from a bank or repay coins in a bank, then related bank will send a record-transaction containing encrypted shares about the borrow-amount (negative number) or repay-amount (positive number) to the blockchain. After that, other banks can confidentially get their shares from the record-transaction, respectively. According to the theory of secret sharing, we know that the shared secret values are always confidential as long as a certain number of banks are honest.

Moreover, when the client wants to borrow coins from a bank, the bank can confidentially get the client's residual loanable-limit by querying other banks. Specifically, a queried bank can add all its shares related to the client to generate a response and then secretly sends the response to the query bank. After receiving responses, the query bank can verify whether the responses are correctly computed by corresponding banks. Then, if the query bank can collect

at least a threshold number of correct responses, then it can recover the client's true residual loanable-limit. Finally, if the residual loanable-limit is a positive number, then the query bank can borrow the client at most the residual loanable-limit number of money. However, if the residual loanable-limit is zero or negative, then the query bank will not lend the client. In this process, the query bank cannot get anything except the residual loanable-limit. Therefore, Loamit can significantly protect the privacy of clients and banks. An overview of Loamit's working process is described in Fig.1. Let $t > \frac{n}{2}$. Next, we take the (t, n) threshold Loamit system as an instance to introduce features of Loamit.

- **Fault-tolerance.** Loamit can work smoothly if at least t banks are active and honest. In other word, Loamit allows that at most $n - t$ banks are malicious, off-line or damaged.
- **Addition homomorphism.** Polynomial secret sharing satisfies additional homomorphism. That is, the sum of shares is the share of the sum of polynomials.
- **Threshold.** A bank can successfully obtain a client's residual loanable-limit if it can collect at least t correct responses.
- **Verifiability or public verifiability.** Sensitive data stored in the blockchain is verifiable or even publicly verifiable. Specifically,
  - Anyone can verify whether a commitment of share or response is correctly computed by the corresponding bank.
  - A bank can verify whether his shares are correctly computed by the corresponding banks.
  - A bank can verify whether received responses are correctly computed by the corresponding banks.
- **Tamper-resistance and credibility** If a transaction has been recorded in the blockchain, it cannot be modified or deleted as well as all publicly verifiable part of the transaction is credible.

- **Privacy.** All amounts of loans and repayments are confidential to banks as long as more than $n-t$ banks are honest. Moreover, a bank cannot get anything from the Loamit system except the residual loanable-limit.

3. Preliminaries

In this section, we will introduce preliminaries of blockchain. *Blockchain, transaction and block* will be shown at first.

3.1. Blockchain, transaction and block

Blockchain [1], which is a "chain" of "blocks", is maintained in a distributed manner by anonymous record-nodes via a selected consensus scheme. Consensus scheme is used to ensure the consistency and tamper-resistance of the blockchain. The record-nodes are connected by a reliable peer-to-peer network.

In the blockchain, the *transaction* is the basic unit, which contains two parts: *transaction header* and *payload*. Transaction header and payload are shown in Table 1. In Loamit, the payload might contain secret or public data that may be used in verification, decryption or computation.

Furthermore, a *block* contains two parts: *block-body* and *block-header*. Specifically, a block-body contains transactions, while a block-header contains: hash value of this block, hash value of the previous block, current Unix time, merkle root of transactions and some other information. The structure of a block is shown in Table 2.

3.2. Consensus scheme

In the process of blockchain generation, time is also divided into epoches. In each epoch, record-nodes will generate a single block belonging to this epoch via a selected *consensus scheme*. This block includes transactions generated in this epoch or previous epoch. Upon verifying the block, all record-nodes will accept the block and add this block as a new one in its local blockchain. After that, all record-nodes join in the next epoch and work for the next block.

Table 1: Format of Transaction

Transaction Header	
Hash	The transaction's hash value
Block number	Block containing the transaction
Order	The transaction's number in the block
Timestamp	Creation time of the transaction
Sender	Sender's ID
Receiver	Receiver's ID
Signature	Sig{the transaction's hash value}
...	...
Payload	
data_1,data_2,$\cdots$,data_n	

Besides, record-nodes are responsible to verify all publicly verifiable data of transactions before the transactions are included in the blockchain. If any publicly verifiable data is invalid, then honest record-nodes will reject corresponding transaction. The invalid transaction then will not be included in the blockchain.

Currently, there are many kinds of consensus schemes. The most popular consensus schemes are Proof-of-work (PoW) [7], Proof-of-stake (PoS) [8] and Practical Byzantine Fault-tolerant (PBFT) [3]. In the three consensus schemes, they have several differences. Firstly, PoW assumes that a majority of computational power is controlled by honest nodes. PoS assumes that a majority of coins is controlled by honest nodes. PBFT assumes that more than $\frac{2}{3}$ of nodes are honest. Secondly, in the PoW-based blockchain, a node controlling more computing power has a bigger probability to generate a block. In the PoW-based blockchain, a node controlling more coins has a bigger probability to generate a block. However, in PBFT-based blockchain, a node controlling more nodes has a bigger probability to generate a block. Finally, the PoW-based blockchain and PoS-based blockchain have a obvious probability to generate a fork. The fork means that there exist multiple valid blocks in an epoch. This will lead to

Table 2: Format of Block

Block Header	
Name	*Description*
Version	Block version number
Hash	The block's hash value
Parent hash	The previous block's hash value
Timestamp	Creation time of the block
Merkle root	The root of Merkle Tree of transactions
...	...
Block Body: Transactions	
Transaction_1,$\text{Transaction}_2 \cdots$,$\text{Transaction}_n$	

a non-single history. However, in a PBFT-based blockchain, there is no fork. Therefore, a PBFT-based blockchain can have a single history.

3.3. Shamir's (t, n) Secret Sharing

Alice wants secretly share a secret value s with n participants, and arbitrary t of the n participants can recover s, but less than t participants cannot. To do this, Alice needs to generate n shares about s, and then secretly send the shares to the n participants, respectively. After that, if someone can collect at least t correct shares, then he can recover the secret value s. Shamir's (t, n) secret sharing (SSS) [9] can accomplish this. Next, we introduce the working process of the SSS.

Firstly, Alice randomly samples a polynomial $f(x)$ of degree t-1 from $\mathbb{F}_p[x]$ as the following polynomial:

$$f(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \cdots + a_1x + s,$$

where s is the secret value, $a_1, \cdots, a_{t-1} \in \mathbb{F}_p$ and $a_{t-1} \neq 0$.

Secondly, let ID_i denote P_i's address. Alice will generate n shares as follows:

$$Share_i = f(ID_i),$$

where i=1, 2, ...,n. Then, Alice secretly sends $Share_1$, $Share_2$, ...,$Share_n$ to the n participants, respectively.

Finally, if someone collects t correct shares, then he can use the *lagrange interpolation* to reconstruct a polynomial. Without loss of generality, let the t shares be $Share_1$, $Share_2$, ...,$Share_t$. He can reconstruct the polynomial $f(x)$ as follow:

$$f(x) = \sum_{i=1}^{t} Share_i \prod_{j=1, j\neq i}^{t} \frac{x - ID_j}{ID_i - ID_j}.$$

Finally, he can get $s = f(0)$.

3.4. Homomorphism of SSS

Let $P_1, P_2, ..., P_n$ be n participants, and ID_i denotes P_i's address. Moreover, $f_1(x)$, $f_2(x)$,...,$f_k(x)$ are k polynomials of degree $t-1$ from $\mathbb{F}_p[x]$. Alice uses the k polynomials to compute all shares to the n participants. After that, we know that $f_1(ID_i)$, $f_2(ID_i)$,...,$f_k(ID_i)$ are all shares of P_i. Importantly, we have $\sum_{j=1}^{k} f_j(ID_i)$ is P_i's share of the following polynomial

$$F(x) = f_1(x) + f_2(x) + ... + f_k(x).$$

Therefore, if someone can collect t correct shares like $\sum_{j=1}^{k} f_j(ID_1)$, $\sum_{j=1}^{k} f_j(ID_2)$,..., $\sum_{j=1}^{k} f_j(ID_t)$, then he can use the *lagrange interpolation* to reconstruct the polynomial

$$F(x) = f_1(x) + f_2(x) + ... + f_k(x).$$

Then, he can get $F(0)$, which is the sum of $f_1(0)$, $f_2(0)$, ..., $f_k(0)$.

4. System Setting and Adversary Model

In this section, we introduce system setting and adversary model. We will describe *Blockchain Network and Cryptographic Keys* used in the system at first.

4.1. Blockchain Network and Cryptographic Keys

Loamit is comprised of record-nodes and banks (light-nodes). Specifically, all record-nodes are connected by a reliable peer-to-peer network, and each bank connects with a certain number of record-nodes. Record-nodes are responsible to maintain the blockchain via Practical Byzantine Fault-tolerance (PBFT) consensus scheme and store the entire blockchain list. Time is divided in to epoches. In an epoch, record-nodes collect and verify transactions sent to the blockchain network, and they record valid transactions their local blocks. By performing the PBFT consensus scheme, some record-node's block become the valid block of the epoch. After that, all record-nodes join in the next epoch to build the next block. However, banks store all block-headers, rather than the entire blockchain list.

Moreover, Loamit uses the Shamir's (t, n)-secret sharing (SSS) [9] to share amounts of loans or repayments. In the implementation of Loamit, (i) secp256k1-based [10] (a elliptic curve) ECDSA [11] is the signature schceme $\mathsf{Sig}(\cdot)$, (ii) secp256k1-based ECIES [12] is the encryption scheme $\mathsf{Enc}(\cdot)$ and (iii) SHA-256 [1] is the hash function $\mathsf{H}(\cdot)$. Besides, we use secp256k1-based point multiplication to compute commitments of shares or responses.

Moreover, a node is honest if it follows all protocol instructions and is perfectly capable of sending and receiving information. While, a node is malicious if it can deviate arbitrarily from protocol instructions. Finally, in a blockchain system, all users communicate with each other via transactions of blockchain, and they only trust messages presented at blockchain.

4.2. Assumptions

In the Loamit system, we have the following assumptions.

- More than $\frac{2}{3}$ of record-nodes are honest. This is the requirement of PBFT.
- In a (t, n)-threshold Loamit system containing n banks, at least $n - t + 1$ banks are honest. This is the security base of SSS.

- Digital signature $\mathsf{Sig}(\cdot)$, encryption scheme $\mathsf{Enc}(\cdot)$ and hash function $\mathsf{H}(\cdot)$ are ideal such that no one can violate $\mathsf{Sig}(\cdot)$, $\mathsf{Enc}(\cdot)$ and $\mathsf{H}(\cdot)$.

5. Loamit

In this section, we introduce how Loamit works. We will describe *transaction and block* used in the system at first.

5.1. Transaction and Block

In Loamit, transactions are divided into 4 types according to payload. They are *limit-transaction, record-transaction, query-transaction and respond-transaction* that can be described by T_{limit}, T_{record}, T_{query} and $T_{respond}$ as follows:

T_{limit}
Transaction Header
Payload
Client ID
Commitments of secret sharing polynomial
Commitments of shares
Encrypted shares

T_{record}
Transaction Header
Payload
Client ID
Commitments of secret sharing polynomial
Commitments of shares
Encrypted shares

T_{query}
Transaction Header
Payload
Client ID

$T_{respond}$
Transaction Header
Payload
$\mathrm{ID}_{T_{query}}$
Commitment of response
Encrypted response

Furthermore, record-nodes may perform two kinds of verifications on transactions. The first one is the *basic verification*, which should be performed on all transactions. Basic verifications contains:

- The transaction should be well-formed.
- The transaction's inputs should have not been used previously.
- The transaction's signature should be valid.

- The sum of input coins should be equal to the sum of output coins.

Except the basic verifications, record-nodes may perform *payload verifications* on limit-transaction, record-transaction and respond-transaction. It means that, in the payloads of limit-transaction, record-transaction and respond-transaction, there are publicly verifiable data that can be verified by record-nodes. If a transaction has presented at the blockchain, then it means that record-nodes have accepted the transaction's publicly verifiable data. Therefore, the transaction's receiver can consider that the transaction's publicly verifiable data is credible. Thus the receiver just needs to perform some other verifications that can be performed only by the receiver. In this way, the most part of verification computations are performed by record-nodes and it helps to decreases bank's verification computations significantly. Table.3 describes verifications of limit-transaction, record-transaction, query-transaction and respond-transaction.

Table 3: Verifications performed by record-nodes on Transactions

Transaction	Basic verification	Payload verification
T_{limit}	✓	✓
T_{record}	✓	✓
T_{query}	✓	
$T_{respond}$	✓	✓

5.2. Construction of Loamit

To clearly introduce the Loamit system, in this subsection, we describe an (t, n)-threshold Loamit system that contains n banks. The symbols used in the paper are shown in Table 4.

Next, we take a client as an example to introduce the working process of Loamit system. C and ID_C denote the client and the client's ID, respectively. Moreover, B_1, B_2, ... ,B_n denote the n banks and $ID_{b,1}$, $ID_{b,2}$, ... ,$ID_{b,n}$ indicate the n banks' IDs. The working process of Loamit are as follows:

Table 4: Symbols of Loamit

Symbol	Description
g	the generator of a cyclic group $\mathbb{G}$
e	the bilinear map, e: $\mathbb{G} \times \mathbb{G} \to \mathbb{G}$.
$\mathbb{F}_p$	the finite field with character p
$ID_{b,i}$	the i-th bank's ID
ID_c	the client's ID
VK	the verification key
$\{pk_{b,i}, sk_{b,i}\}$	the i-th bank's key pair
$\{pk_c, sk_c\}$	the client's key pair
$\{Share_{i,j}\}$	the i-th bank's share of the j-th transaction
$Resp_i$	the i-th bank's respnse

- **Step 1: Build an account.** C wants to handle a business in B_1. The business is C's first bank business in the n banks. Then the bank builds an account for C. To do this, B_1 randomly samples a polynomial $f_0(x)$ of degree t-1 from $\mathbb{F}_p[x]$ as the following polynomial:

$$f_0(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \cdots + a_1x + limit,$$

where $limit, a_1, \cdots, a_{t-1} \in \mathbb{F}_p$ and $a_{t-1} \neq 0$. Moreover, the $limit$ is the upper loanable-limit of C and $limit$ is a positive number. Then B_1 generates shares $Share_{0,1}$, $Share_{0,2}$, $\cdots$, $Share_{0,n}$ as follows:

$$Share_{0,i} = f_0(ID_{b,i}),$$

i from 1 to n. Then B_1 generates commitments of efficiencies of $f_0(x)$ as follows:

$$CM_{a_{t-1}} = g^{a_{t-1}}, CM_{a_{t-2}} = g^{a_{t-2}}, ..., CM_{limit} = g^{limit}.$$

After that, for i from 1 to n, B_1 produces commitment of $Share_i$ as follows:

$$CM^S_{0,i} = g^{Share_{0,i}}$$

where g is an elliptic curve base point. For i from 1 to n, B_1 encrypts $Share_{0,1}$, $Share_{0,2}$, $\cdots$, $Share_{0,n}$ into $C^S_{0,1}$, $C^S_{0,2}$, $\cdots$, $C^S_{0,n}$ follows:

$$C^S_{0,i} = E_{pk_i}(Share_{0,i}),$$

where pk_i is the B_i's public key, i from 1 to n. Then, B_1 generates a limit-transaction T_{limit} as follows:

T_{limit}			
Transaction Header			
Payload			
ID_C			
$CM_{a_{t-1}}$	$CM_{a_{t-2}}$	$\cdots$	CM_{limit}
$CM^S_{0,1}$	$CM^S_{0,2}$	$\cdots$	$CM^S_{0,2}$
$C^S_{0,1}$	$C^S_{0,2}$	$\cdots$	$C^S_{0,n}$

After that, B_1 sends the T_{limit} to blockchain network.

- **Step 2: Record-nodes verify limit-transactions.** Honest record-nodes will verify all new limit-transactions before appending them on the blockchain. Specifically, when an honest record-node receives the limit-transaction generated in **Step 1**, the record-node will verify the transaction's commitments of shares as follows:
  - Get $\{CM_{a_{t-1}}, \cdots, CM_{a_1}, CM_{limit}, CM^S_{0,1}, \cdots, CM^S_{0,n}\}$ from the limit-transaction.
  - Verify whether the following equation always holds for i from 1 to n.

$$\begin{aligned}&(CM_{a_{t-1}})^{ID^{t-1}_{b,i}} \cdots (CM_{a_1})^{ID_{b,i}}(CM_{limit})\\ &= CM^S_{0,i}\end{aligned}$$

  - If the above equation always holds for i from 1 to n, then the record-node accepts the limit-transaction, otherwise the record-node rejects it.

- **Step 3: Other banks verify limit-transaction.** For i from 1 to n, when the bank B_i sees the T_{limit} in the blockchain, B_i will verify its share as follows:

– Decrypt $C^S_{0,i}$ into $Share'_{0,i}$.

– If

$$CM^S_{0,i} = g^{Share'_{0,i}},$$

then B_i accepts that its share included in the limit-transaction is valid, otherwise B_i rejects the share.

- **Step 4: Record amounts of loans and repayments.** If C borrows N_{bor} coins from (or repay N_{rep} coins to) B_2, then B_2 will generate a record-transaction for the loan business of C. The generation process of record-transaction is same as the limit-transaction. Merely, if it is a loan business, then the secret value of record-transaction is a negative number. In contrast, if it is a repay business, then the secret value of record-transaction is a positive number. After that, a record-transaction can be described as follows:

T_{record}			
Transaction Header			
Payload			
ID_C			
$CM_{a'_{t-1}}$	$CM_{a'_{t-2}}$	$\cdots$	CM_{secret}
$CM^S_{p_1,1}$	$CM^S_{p_1,2}$	$\cdots$	$CM^S_{p_1,n}$
$C^S_{p_1,1}$	$C^S_{p_1,2}$	$\cdots$	$C^S_{p_1,n}$

Then, B_2 sends the record-transaction to blockchain network. After that, record-nodes and other banks will verify the record-transactions as same as they do in Step 2 and Step 3.

- **Step 4.5: Client check shared amounts.** After the client's record-transactions have been recorded in the blockchain, he can check whether his business amounts are correctly recorded in the blockchain. For instance, a commitment of the client's loan amount is $CM_{N_{loan}}$. In fact, the client knows the real loan amount N'_{loan} corresponding to $CM_{N_{loan}}$. Therefore, if the following equation holds, then he will consider that the

commitment $CM_{N_{loan}}$ is correctly computed.

$$CM_{N_{loan}} = g^{N'_{loan}}$$

- **Step 5: Query.** If C wants to apply a loan business in a bank. For instance, the bank is B_{qeury}. Then, B_{qeury} needs to check C's residual loanable-limit before lending coins to C. Let $N_{residual}$ denote C's residual loanable-limit at this moment. If $N_{residual}$ is zero or negative, then C cannot borrow any coins from the bank. However, if $N_{residual}$ is a positive number, then C can borrow at most $N_{residual}$ coins from B_{qeury}. To realize this function of query, B_{qeury} sends a query-transaction to blockchain network for getting C's residual loanable-limit. The query-transaction is described as follows:

T_{query}
Transaction Header
Payload
ID_C

- **Step 6: Respond.** After the T_{query} sent by B_{qeury} has presented in the blockchain, if a bank wishes to respond the query, then it will generate a response according to the T_{query}. Then the bank will secretly send his response to B_{qeury} via transaction. Finally, if B_{qeury} may collet at least t responses correctly computed by corresponding banks, then B_{qeury} can recover the C's correct residual loanable-limit. For instance, B_3 wishes to respond the T_{query}. Let $Resp_{b,3}$ be the response generated by B_3. B_3 can generate $Resp_{b,3}$ as the following equation with all B_3's shares related to ID_C:

$$Resp_{b,3} = Share_{0,3} + Share_{1,3} + ... + Share_{k,3},$$

where $Share_{0,3}$, $Share_{1,3}$,...,$Share_{k,3}$ are all B_3's shares related to C. Then, B_3 generates a commitment $CM_{Resp_{b,3}}$ of the $Resp_{b,3}$ as follow:

$$CM_{Resp_{b,3}} = g^{Resp_{b,3}},$$

After that, B_3 encrypts $Resp_{b,3}$ into $C_{Resp_{b,3}}$ as follow:

$$C_{Resp_{b,3}} = E_{pk_{query}}(Resp_{b,3}),$$

where pk_{query} is the bank B_{qeury}'s public key. Then, B_3 generates a respond-transaction as follow:

$\mathrm{T}^{3}_{respond}$
Transaction Header
Payload
ID_C $CM_{Resp_{b,3}}$ $C_{Resp_{b,3}}$

Finally, B_3 sends the respond-transaction $T^3_{respond}$ to blockchain network. Similarly, other banks may generate and send similar respond-transactions to blockchain network. For instance, B_4 may send a respond-transaction as follow:

$\mathrm{T}^{4}_{respond}$
Transaction Header
Payload
ID_C $CM_{Resp_{b,4}}$ $C_{Resp_{b,4}}$

- **Step 7: Record-nodes verify respond-transactions.** After receiving a new respond-transaction, a record-node will verify the validation of the transaction's commitment of response. For example, a record-node receives $T^4_{respond}$ which is B_4's respond-transaction. The record-node will process it as follows:
  - Collect all commitments of shares of B_4 related to ID_C. Let the commitments of shares be $CM_{0,4}$, $CM_{1,4}$,..., $CM_{t,4}$.
  - Extract the commitment $CM_{Resp_{b,4}}$ from $T^4_{respond}$.
  - If the follow equation holds,

    $$CM_{0,4}CM_{1,4}...CM_{t,4} = CM_{Resp_{b,4}},$$

    then the record-node will accept the commitment of B_4's response, otherwise the record-node rejects it.

- **Step 8: Recover.** B_{qeury} is the query bank. If B_{qeury} can collect at least t correct responses, then B_{qeury} can correctly recover the real residual loanable-limit of C. For instance, $T^1_{respond}$, $T^2_{respond}$, $\cdots$, $T^t_{respond}$ have presented at the blockchain. It means that all publicly verifiable data of the transactions is valid. Therefore, B_{qeury} just needs to decrypt all encrypted responses. After decryptions, B_{qeury} get t un-encrypted responses that are $Resp'_{b,1}$, $Resp'_{b,2}$,..., $Resp'_{b,t}$. Then, for i from 1 to t, if the following equation holds, then B_{qeury} accepts $Resp'_{b,i}$, otherwise B_{qeury} rejects it.

$$CM_{Resp_{b,i}} = g^{Resp'_{b,i}},$$

If the t responses are all accepted, then B_{qeury} may use the *lagrange interpolation* to reconstruct a polynomial as follow:

$$\tilde{F}(x) = \sum_{i=1}^{t} Resp'_{b,i} \prod_{j=1, j\neq i}^{t} \frac{x - ID_{b,j}}{ID_{b,i} - ID_{b,j}}.$$

Finally, B_{qeury} calculates $\tilde{F}(0)$ that is real residual loanable-limit of C.

6. Performance Evaluation

In this section, we give a Loamit's performance evaluation that may be broken into three parts. The first part studies the processing time of cryptographic and mathematic computations. The time of processing transactions is researched in the second part. The last part further demonstrates the processing time of blocks when different transactions are sent to the blockchain network. The section starts with the prototype system setting.

6.1. Prototype System Setting

Loamit's efficiency mainly depends on the blockchain platform, computer platform and performance of cryptographic schemes. For instance, in this paper, we use the Ethererum blockchain as the platform. Specifically, in the Ethereum blockchain, (i) a block can contains at most 62360-byte transactions, (ii) its average block interval is about 15 s and (iii) its transaction's payload contains at

most 1014-byte data. Consequently, Loamit's efficiency is significantly limited by the blockchain plarform. Therefore, if we use a more efficient blockchain platform, then we might get a better throughput.

In the prototype system, we implement a (2,3)-threshold Loamit prototype system that contains three banks. Let the three banks be B_1, B_2 and B_3. If a bank can collect two correct responses from two banks (including the response generated by the query bank itself), then the query bank can obtain the correct residual loanable-limit of a client. We use laptops and virtual machines to perform the prototype system. Specifically, our laptop's configuration is described as follows: the Intel i5-5300 CPU with 2.30GHz, 4GB memory, Ubuntu 16.04. In the local area network, we deploy a local blockchain via go-ethereum that is a Go implementation of the Ethereum protocol [13]. In the blockchain network, we deploy four record-nodes (miners), and we use transaction simulator [13] to simulate banks to generate and send transactions. Moreover, we record Loamit system's key data in the transaction's payload.

Additionally, Ethererum has a embedded signature scheme that is the ECDSA based on the secp256k1 elliptic curve [10]. For convenience, we use the scheme to sign messages. Besides, to encrypt sensitive data recorded in the payloads of limit-transaction, record-transaction and respond-transaction, we use ECIES based on secp256k1 to encrypt some data via receiver's public key, where the cipher block has a length of 64 bytes. It results in that each encrypted data has a length of 64 bytes. Moreover, the encrypted data can be decrypted only by the corresponding receivers since only the receiver has the corresponding private key. Furthermore, we utilize the secp256k1-based point multiplication to commit sensitive data.

6.2. Processing Time of Cryptographic Schemes

Generally, the time cost of performing cryptographic schemes will have a certain degree of influence on the time of processing transactions. Therefore, in the sub-section, we study the time cost of cryptographic schemes.

For each of point multiplication, point addition, field addition, field multiplication, encryption, decryption, signing and verifying signatures, we perform 1000 experiments to obtain their average time cost, and the average time cost is described in Table 5.

Table 5: Average Time cost of cryptographic schemes

Scheme	Time cost
Secp256k1-based Point Mul	596.113 μs
Secp256k1-based Add	2.328 μs
Field Add	0.071 μs
Field Mul	0.531 μs
Secp256k1-curve ECDSA Sign	4.425 ms
Secp256k1-curve ECDSA Verify Sig	9.137 ms
Secp256k1-curve ECIES Encryption	8.745 ms
Secp256k1-curve ECIES Decryption	4.367 ms
Block Interval	15.3 s

6.3. Generate Transactions

In the Loamit system, different transactions may have different generation time. Consequently, in the sub-section, we study generation time of transactions in the prototype system. We discuss the limit-transaction and record-transaction at first.

In the prototype system, we implement a (2,3)-threshold Loamit instance. Therefore, the payload of the T_{limit} or T_{record} includes a client's ID, two commitments of efficiencies of polynomial, three commitments of shares and three encrypted shares as mentioned at section 5.2. According to the last sub-section, these data have a length of 544 bytes. We know that the payload of a transaction, in the Ethererum blockchain, can include at most 1014 bytes. Therefore, a normal transaction is enough to record all those data. Specifically, the limit-

transaction and record-transaction can be described as follows:

T_{limit}		
Transaction Header		
Payload		
ID_C	CM_{a_1}	CM_{limit}
$CM^S_{0,1}$	$CM^S_{0,2}$	$CM^S_{0,3}$
$C^S_{0,1}$	$C^S_{0,2}$	$C^S_{0,3}$

T_{record}		
Transaction Header		
Payload		
ID_C	$CM_{a'_1}$	CM_{secret}
$CM^S_{p1,1}$	$CM^S_{p1,2}$	$CM^S_{p1,3}$
$C^S_{p1,1}$	$C^S_{p1,2}$	$C^S_{p1,3}$

In the implementation, the sizes of query-transaction and respond-transaction are fixed. Specifically, they can be described as follows:

T_{query}
Transaction Header
Payload
ID_C

$\mathrm{T}^k_{respond}$		
Transaction Header		
Payload		
ID_C	$CM_{Resp_{b,k}}$	$C_{Resp_{b,k}}$

In the above table, k=1, 2 or 3.

In our experiments, sizes of the four transactions' payloads are shown in Table 6. For each of T_{limit}, T_{record}, T_{query} and $T_{respond}$, we generate 1000 transactions in order to obtain their average generation time cost, and their average time cost are shown in Table 7.

Table 6: Payloads of Transactions Used in the Prototype System

Payload	Size
Payload of T_{limit}	544 bytes
Payload of T_{record}	544 bytes
Payload of T_{query}	32 bytes
Payload of $T_{respond}$	160 bytes

6.4. Verify transactions

In the prototype system, before a transaction is appended at the blockchain, record-nodes must verify the transaction. Specifically, record-nodes verify all publicly verifiable data of the transaction. Moreover, if a transaction has

appeared at the blockchain, then the transaction's publicly verifiable data is credible. Consequently, banks do not have to verify the transaction's publicly verifiable data. In this way, it significantly reduces verifying computations of banks. In the sub-section, we study transactions' verification time cost. All publicly verifiable data of transactions are summarized as follows:

- All transactions' signatures are publicly verifiable data that can be verified by record-nodes. Therefore, if a transaction has appeared at the blockchain, then it means that the transaction's signature is credible, and others donot need to verify the signature.

- Except signatures, the payloads of limit-transaction, record-transaction and respond-transaction contain public verifiable data that can be verified by record-nodes. Specifically, they are commitments of shares and commitments of responses.

In this way, a bank just needs to verify some key data that can be verified only by itself.

For each of T_{limit}, T_{record}, T_{query} and $T_{respond}$, we generate 1000 transactions, and then obtain their average verifying time cost, which are shown in Table 7. Specifically, in Table 7, S is a signing computation, V denotes a signature verification, PM describes a point multiplication on the ECC, PA is a point addition on the ECC, E is a encryption, D denotes a decryption, FM describes a field multiplication and FA is a field addition. For instance, "2PM+3PA+1V+1E" denotes that the corresponding computations contain 2 point multiplications, 3 point additions, 1 signature verification and 1 encryption.

6.5. Blockchain Performance Evaluation

We run our Loamit prototype system on the Ethereum blockchain. After generating a certain number of blocks, the block interval tends to be stable. That is, generating 1000 blocks takes about 4.3 hours. In other words, generating a block takes about 15.2 s on average. Furthermore, in the Ethereum blockchain, a block can record transactions of at most 62360 bytes, a transaction with an

Table 7: Average Time cost of processing transactions

Operation on transaction	Computations	Time cost
Bank generates a T_{limit}	1S+5PM+3E	34.465 ms
Bank generates a T_{record}	1S+5PM+3E	34.576 ms
Bank generates a T_{query}	1S	5.247 ms
Bank generates a $T_{respond}$	1S+kFA+1PM+1E	14.588 ms
Record-node verifies a T_{limit}	1V+3PM+3PA	10.932 ms
Record-node verifies a T_{record}	1V+3PM+3PA	10.872 ms
Record-node verifies a T_{query}	1V	9.131 ms
Record-node verifies a $T_{respond}$	1V+kPA	9.149 ms
Bank verifies a T_{limit}	1D+1PM	4.963 ms
Bank verifies a T_{record}	1D+1PM	4.876 ms
Bank verifies a $T_{respond}$	1D+1PM	4.398 ms
Bank recovers the result	4FA+4FM	2.408 μs

In the table, S is a signing computation, V denotes a signature verification, PM describes a point multiplication on the ECC, PA is a point addition on the ECC, E is a encryption, D denotes a decryption, FM describes a field multiplication and FA is a field addition. For instance, "2PM+3PA+1V+1E" denotes that the corresponding computations contain 2 point multiplications, 3 point additions, 1 signature verification and 1 encryption.

empty payload is of 308 bytes and a transaction's payload can record data of at most 1014 bytes. Therefore, a transaction's size should be from 308 bytes to 308+1014=1322 bytes.

According to Table 6 and above contents, in our implementation, any transaction's size can be calculated. Transactions' sizes are shown at Table 8. A block can record transactions of at most 62360 bytes. Therefore, if a block only record identical transactions, then the number of recorded transactions has a limit. The limits are described in the Fig.2.

In our experiments, because different transactions have different significance, so more significant transactions should be processed more early than less significant transactions. In the Ethererum blockchain, record-nodes (miners) earlier process the transaction with more fee. Therefore, we set different transaction-

Table 8: Transactions' sizes in Our Simulation

Transaction	Size
T_{limit}	852 bytes
T_{record}	852 bytes
T_{query}	340 bytes
$T_{respond}$	468 bytes

s having different transaction fees. When different transaction are pending in record-nodes transaction pool, transactions with more fees will be recorded earlier. In the Loamit prototype system, the limit-transaction and record-transaction are the base of later transactions. Therefore, they have the first priority. For quickly responding bank's query, we set that query-transaction has the second priority and respond-transaction has the third priority. In our experiments, their transaction fees are shown at Table 9. In our experiments,

Table 9: Transaction fee

Transaction	Transaction fee
T_{limit}	0.002 ETH
T_{record}	0.002 ETH
T_{query}	0.0015 ETH
$T_{respond}$	0.001 ETH

after a query-transaction has appeared in the n-th block of the blockchain, corresponding respond-transaction will appear in the $n + 1$-th block quickly. Therefore, the time from sending query-transaction to recovering desired residual loanable-limit of a client may be 15 s to 30 s.

7. Conclusion

In this paper, we propose a blockchain-based residual loanable-limit query system, called Loamit. The Loamit system is composed of a certain number of banks. After opening an account for a new client, the related bank will generate an initial loanable-upper-limit to the client by sending a limit-transaction to the blockchain at the same time. Moreover, all amounts of loans or repayments are confidentially shared among banks by using verifiable secret sharing. The shared amounts are always confidential as long as a certain number of banks are honest. Besides, any bank can only inquire the residual loanable-limit of a client, rather than details of loans and repayments. Therefore, this method significantly protects the privacy of clients and banks. Furthermore, the loamit system can smoothly work if a threshold number of banks are active and honest. Thereby, the proposed system has a property of fault-tolerance. Additionally, most of data is verifiable or even publicly verifiable. Consequently, malicious banks can be checked out by verifying received data like shares and responses. Finally, we deploy the Loamit on the Ethereum blockchain and give a various performance evaluation to the proposed system.

Acknowledgment

This work is supported by the National Key Research and Development Program (No. 2016YFB0800602) and the National Natural Science Foundation of China (NSFC) (No. 61502048).

References

[1] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system (2008).

[2] M. Abramowicz, Cryptocurrency-based law (2016).
URL http://arizonalawreview.org/pdf/58-2/58arizlrev359.pdf

[3] M. Castro, B. Liskov, Practical byzantine fault tolerance and proactive recovery, ACM Trans. Comput. Syst. 20 (4) (2002) 398–461. doi:10.

1145/571637.571640.
URL http://doi.acm.org/10.1145/571637.571640

[4] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, Zerocash: Decentralized anonymous payments from bitcoin, in: 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014, 2014, pp. 459–474. doi:10.1109/SP.2014.36.
URL https://doi.org/10.1109/SP.2014.36

[5] S. Noether, Ring signature confidential transactions for monero, IACR Cryptology ePrint Archive 2015 (2015) 1098.
URL http://eprint.iacr.org/2015/1098

[6] K. Constantinides, S. Plaza, J. A. Blome, B. Zhang, V. Bertacco, S. A. Mahlke, T. M. Austin, M. Orshansky, Bulletproof: a defect-tolerant CMP switch architecture, in: 12th International Symposium on High-Performance Computer Architecture, HPCA-12 2006, Austin, Texas, February 11-15, 2006, 2006, pp. 5–16. doi:10.1109/HPCA.2006.1598108.
URL https://doi.org/10.1109/HPCA.2006.1598108

[7] C. Dwork, M. Naor, Pricing via processing or combatting junk mail, in: Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings, 1992, pp. 139–147. doi:10.1007/3-540-48071-4_10.
URL https://doi.org/10.1007/3-540-48071-4_10

[8] K. Sunny, S. Nadal, The theory of a general quantum system interacting with a linear dissipative system, self-published paper (2012).

[9] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613. doi:10.1145/359168.359176.
URL http://doi.acm.org/10.1145/359168.359176

[10] D. J. Bernstein, T. Lange, Safecurves: choosing safe curves for elliptic-

curve cryptography (2013).
URL http://safecurves.cr.yo.to

[11] D. Johnson, A. Menezes, S. A. Vanstone, The elliptic curve digital signature algorithm (ECDSA), Int. J. Inf. Sec. 1 (1) (2001) 36–63. doi:10.1007/s102070100002.
URL https://doi.org/10.1007/s102070100002

[12] N. P. Smart, The exact security of ECIES in the generic group model, in: Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings, 2001, pp. 73–84. doi:10.1007/3-540-45325-3_8.
URL https://doi.org/10.1007/3-540-45325-3_8

[13] A. Schoedon, A. A. Fischer, Go-ethereum.
URL https://github.com/ethereum/go-ethereum

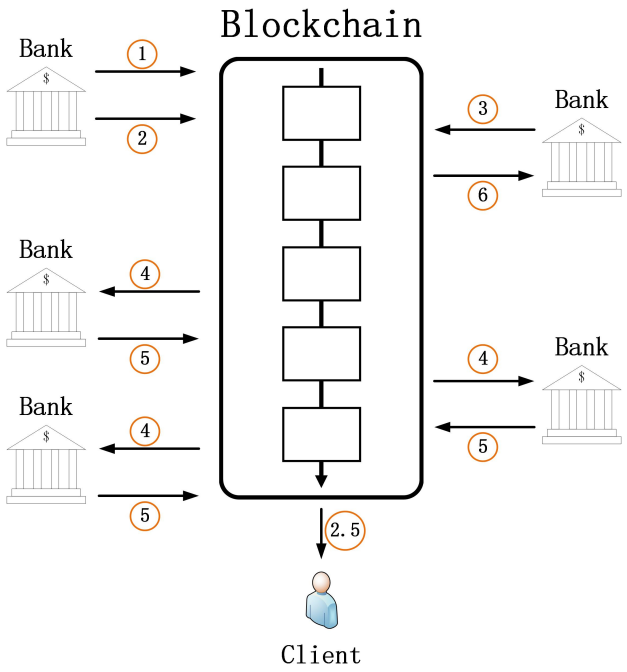

Figure 1: An overview of Loamit

Step 1: After opening an account for a client, a bank sends a limit-transaction to the blockchain for the client. Step 2: After businesses of loans and repayments of the client, the bank of Step 1 sends loan-transactions and repay-transactions to blockchain. Step 2.5: The client can verify whether his data are correctly processed by the corresponding bank. Step 3: If another bank wants to know the client's remaining limit, it sends a query-transaction to blockchain. Step 4: Other banks see the query-transaction and download the limit-transaction, loan-transactions and repay-transactions corresponding to the client. Step 5: Active banks send respond-transactions to blockchain. Step 6: The query bank collects respond-transactions and then recovers the loan remaining limit.

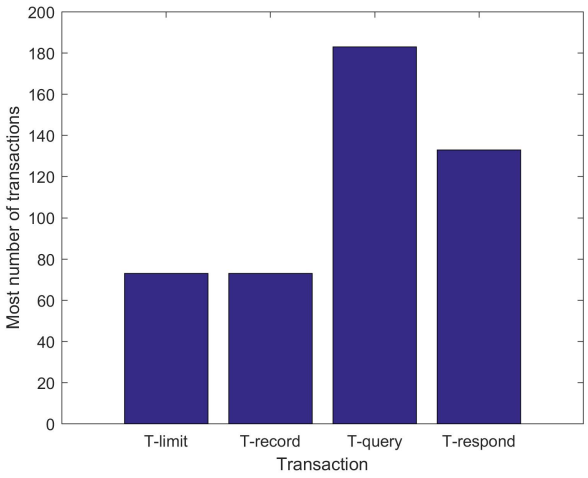

Figure 2: The most number of same transactions recorded in a block.