The University of Sydney

School of Information Technology

Thesis of Master of IT

Secure Unforkable Blockchain Wallet

Author:
Lin Han

Supervisor:
Dr. Vincent Gramoli

November 17, 2017


SIDERE MENS EADEM MUTATO

Special thanks to Dr. Vincent Gramoli and everyone in CSRG of the University of Sydney who provides generous help during my research

Contents

1 Introduction

From the time when Block 0 of Bitcoin blockchain, the Genesis Block, is created at 18:15:05 GMT on January 3rd, 2009, the words "cryptocurrencies" and "Blockchain" become one of the most popular topics in information technology fields. The "decentralized" and "anonymous" nature of cryptocurrencies overcomes the weakness of traditional *trust-based* electronic payments who relies heavily on trusted third-party financial institutions. It's cryptography-based nature also make sure its security in some sense. The *cryptographic proof-of-work* of bitcoin enables reliable transaction between two parties directly[15]. Through almost 10 years development, cryptocurrencies turns out to be a large family and can be accessed via desktop, laptop, or even mobile devices like smart phones.

Though bitcoin gives a practical solution on traditional *double spending* problem in digital currencies, this doesn't mean that it is secure in all aspects. One possible issue is bitcoin blockchain's forkable feature. Actually, other cryptocurrencies allowing forkable chains all suffer from the very same issue. It will become a problem for wallet users if the nodes they are contacting is hijacked. In this sense, unfokable blockchain is proven to overcome this shortcommings[5]. To consider in another aspect, another possible solution to secure transaction is to adopt mechanism like *zero knowledge contingent payment* which are released if and only if some knowledge is disclosed by the payee and to do this in a trustless manner where neither the payer or payee can cheat[20][19]. Whilst there are several theoretical discussion and practices in a variety of contexts, this paper will concentrate on how to develop a secure cryptocurrency wallet on the basis of unforkable blockchain and enable secure trades between parties by the help of zero knowledge contingent payments.

In this project, I intend to contribute in two separated directions: one is to develop a secure mobile client for Red Belly Blockchain - a practice to deploy unforkable blockchain onto modern mobile devices; the other is to explore the way to enable zero knowledge contingent payments within Red Belly Blockchain network. The primary goal of this research is to examine the feasibility of real-life secure application for unforkable blockchain like Red Belly Blockchain.

In the following report, related works will be given first. Essential concepts and problems will be addressed in this part. After the literature review, the methodologies and proposed contribution is given. After that, detailed

report on implementation of secure wallet and zero knowledge contingent payments will be shown. Last but not the least, current outcomes of this project will be discussed. And finally, a conclusion as well as possible improvements and future works will conclude this paper.

2 Literature Review

2.1 Bitcoin

Bitcoin is the first decentralized digital currency as well as a digital payment system. The whole system is peer-to-peer based, whose transactions are between users directly without participation of intermediary. Transactions are verified by network nodes - known as *mining* - and recorded in a public distributed ledger called *blockchain*.

2.1.1 Blockchain

Blockchain is the way how Bitcoin keeps its public ledger within its peer-to-peer network. To some extent, blockchain is a peer-to-peer distributed timestamp server. The ultimate goal of this design is to solve *double-spending* problems and prevent modification of transaction records[15].

Each full node in the Bitcoin network keeps a full copy of the blockchain, in which all blocks validated by this particular is stored. When several nodes within the network independently arrive at identical blockchains, they are considered to be in *consensus*. As its name suggests, a blockchain is a digital chain of blocks, where a timestamp, a nonce, and a Merkle Tree is stored. The blocks are chained cryptographically using hash. In detail, each block contains the hash of its previous block ,finally leading to the Genesis Block. Any modification on blocks in the chain would violates all subsequent hashes, which is vital for consistency of the ledger. Figure 1 shows part of a blockchain.

However, computing a hash is expensive. This truth enables *proof-of-work* in bitcoin network.

2.1.2 Proof of Work

According to blockchains' feature, a huge amount of computation is required in the generation of each block. Meanwhile, there is a *proof-of-work* mech-

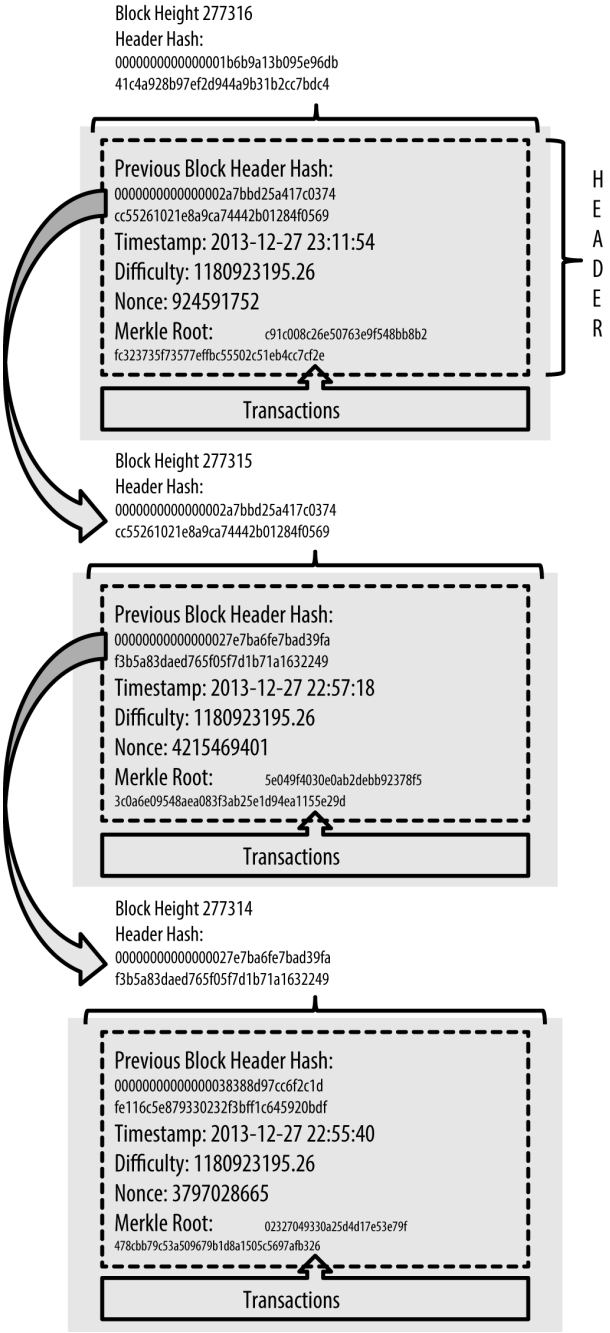

Figure 1: Blockchain

anism to make the distributed timestamp server work and determine representation in majority decision making. Especially, when there are multiple chains (forks), consensus rules will pick up the longest chain, which contains the most proof of work during its generation[15].

In this way, any malicious changes on previous blocks would violate its following blocks. That is to say, hacker with huge computing power can hijack the blockchain if he can generate the longest chain from the block he hacks. In turn, he has to own more than half of the computing power within the whole blockchain network[17].

2.1.3 Contracts

There are distributed contracts in Bitcoin transactions for agreement enforcement, which provides another way to formalize and guarantee agreements rather than traditional court system. Examples include Escrow, Micropayment channels and CoinJoin.

Some of the contracts can be implemented in Bitcoin Script, especially the zero knowledge contingent payments in Bitcoin is achieved with support of bitcoin scripts. However the Red Belly Blockchain doesn't have a robust scripting language like Bitcoin does.

2.2 Ethereum

2.2.1 Previous Work of Bitcoin

Bitcoin provides a protocol allowing weak implementation of *smart contracts*. However, several limitations exists in Bitcoin's scripting language:

1. **Not Turing-Completeness** - Bitcoin scripts lacks loops.
2. **Lack of States** - UTXOs scripts is only for one-off contracts.
3. **Blindness of Blockchain** - Bitcoin scripts cannot access blockchain data.
4. **Blindness of Value** - Bitcoin either consumes the entire UTXO or none of it

2.2.2 Rationale

Ethereum implements a blockchain with Turing-complete scripts, states, value awareness and blockchain awareness, which enables development of smart contracts, and even new protocols[21].

2.3 Balance Attack

As the previous review mentioning, to attack a blockchain, or specifically to rewrite the content of a block, the hacker should have more than half of the computing power of the whole blockchain network which is almost unfeasible in real world. In particular, by delaying the propagation of blocks in Bitcoin system, the hacker can in result delay the growth of the longest branch of the system. In other word, he can then hijack the blockchain even without a large amount of computing power. Ethereums' "Blockchain 2.0" somehow fixes this problem, but there is still other possible method against forked blockchain. One practice is the **Balance Attack** [7][17][16]against *proof-of-work* blockchain systems.

To achieve a balance attack within the blockchain network, the attacker should divide the network into subgroups of similar mining power by cutting off their communications. During this down time, the attacker issues transaction in the *transaction group*, and mine blocks in the *block group* simultaneously. This action only ends when it comes to the point where the tree of the block subgroup outweighs the tree of the transaction group, which is with high possibility. The balance, in result, can leverage the *GHOST* protocol that accounts for sibling or uncle blocks to determine on a chain of blocks. This strategy allows the attacker to mine a branch regardless of the rest of the network so that he can influence the branch determination process while merging[17]. The process is as shown in Figure 2.

2.4 Unforkable Blockchain

2.4.1 Byzantine Consensus Problem

The *Byzantine Consensus Problem* refers to the *Byzantine General's Problem* proposed by Leslie Lamport, Robert Shostak and Marshall Pease in 1982. The problem is complicated by the presence of traitorous generals who may not only cast a vote for a suboptimal strategy, they may do so selectively. All the votes and results are simplified to attack or retreat. The problem is

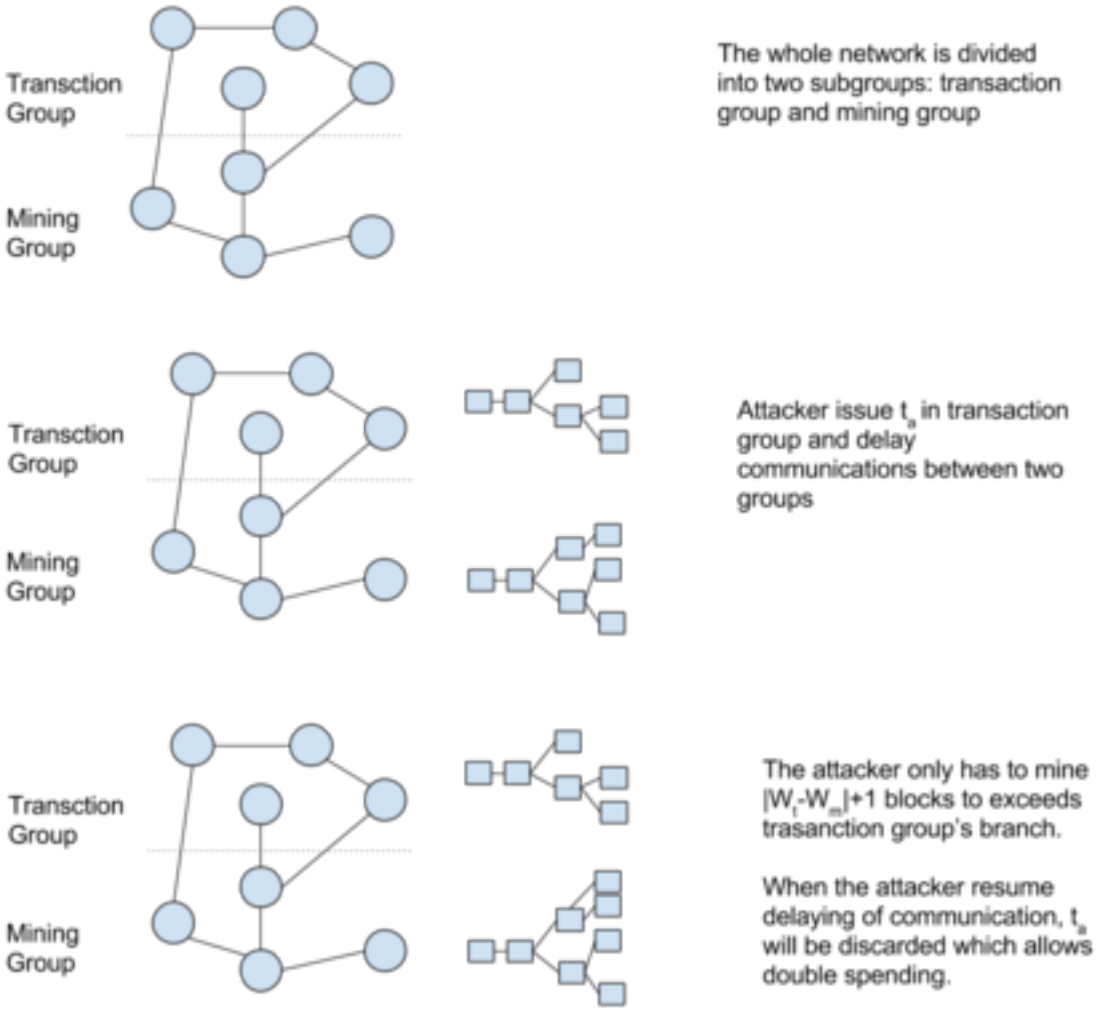

Figure 2: Balance Attack

complicated further by the generals being physically separated and having to send their votes via messengers who may fail to deliver votes or may forge false votes[11].

In computer science area, typically computers or participants in network are mapped to generals and links between them are mapped to messengers.

2.4.2 Traditional Blockchain Byzantine Consensus Problem

The *proof-of-work* of Bitcoin blockchain is the primary solution to *Byzantine Consensus Problem*.

In detail, a model of Bitcoin network can be built upon the classic Byzantine Consensus Problem. The distributed system is the alliance of generals, in which the upper bounds on the delay of communicating and decision-making is unknown.

However, regarding to our previous discussion on *Balance Attack*, attackers can still disrupt the consensus system to beat the Bitcoin Byzantine Consensus[8]. Because there is no guarantee that the decided value is proposed by a valid process.

2.4.3 Democratic Byzantine Consensus

Democratic Byzantine Fault Tolerance is a system specially tailored for consortium blockchains based on *Binary Byzantine Consensus*. In *Binary Byzantine Consensus*[14], each trusted participant issue proposal with either 0 or 1 and decides that the final agreement such that:

1. No pair of trusted participants have different decision.
2. Every trusted participant decides
3. If all correct participant propose the the same value, then no other value can be deiced

In safe Democratic Byzantine Fault Tolerance[5], a mechanism called binary broadcast for binary Byzantine Consensus system is adopted. To conclude, there are four aspects that is strictly followed within DBFT:

1. **Binary Value Obligation** - if t+1 correct BV-broadcast v, then v is eventually added to the set binary values of all correct process.

2. **Binary Value Justification** - if p_i is correct and and v in $binary - values_i$ then v was broadcasted by a correct process.

3. **Binary Value Uniformity** - if v is added to $binary - value_i$ of correct p_i, then eventually v will be in $binary - value_j$ for all correct p_j.

4. **Binary Value Termination** - eventually $binary - value$ of correct p_i is not empty.

2.5 The Red Belly Blockchain

Red Belly Blockchain is new blockchain relying on the Democratic BFT. The *Genesis Block* of this blockchain contains the initial information as well as a list of n participants. All accesses of external nodes requires these participants as the middleware. In this case, a transaction is regarded as committed if $t + 1$ participating nodes agreed so it can be written into next block.

The performance of this new blockchain is quite good. It can achieve more than 400 transactions per second and with great scalability up to 90 server nodes.

2.6 Zero Knowledge Contingent Payment

2.6.1 Bitcoin ZKCP

Zero Knowledge Contingent Payment in Bitcoin is first proposed by Gregory Maxwell in 2011 on Bitcoin Wiki[13]. The basic process can be illustrated by an example[3]:

> Alice is a fan of Sudoku puzzle. However, there is a puzzle that she is trying days but in vain. She gives up and broadcasts a message within a fan group proclaiming that "I will pay whoever solves this puzzle." Bob see the broadcast, solves it, and want to sell the result to Alice. The problem is either Alice or Bob is willing to be the first person to give out what they have.
>
> To solve this dilemma, Alice and Bob goes for a Bitcoin, which allows one to issue a transaction and also specify the conditions to be met in order to claim the transaction. In this case, Alice

propose a payment transaction to blockchain that includes encoded Sudoku puzzle and the rules. Whoever solves the puzzle is able to get the fund.

Using the Bitcoin ZKCP protocol, Bob knows a solution s and encrypts the solution using a key k such that $Enc_k(s) = c$ and he computes y such that $SHA256(k) = y$. He then send the key k and c to Alice together with a zero knowledge proof that c is an encryption of s under the key k and that $SHA256(k) = y$. Once Alice has verified the proof, she creates a transaction to the blockchain that pays Bob n bitcoins, and says that Bob can only claim the coins if he provides the value k' such that $SHA256(k') = y$. Bob then published k and claims the fund. In this way, Alice learns k can decrypt c so that she knows the solution s

Specifically in Blockchain, the ZKCP protocol now has several practices rather than theory. One is ZK-SNARK protocols allowing for the practical implementation of the necessary proofs[10][2][4].

2.6.2 Zero Knowledge Contingent Payment without Scripts

However, all implemented ZKCP protocols now are based on scripting language in traditional forkable blockchain[1]. Besides, there is no convincing data of its performance on modern mobile devices[22].

3 Motivation

From the above review of cryptocurrencies and blockchain contexts, one can conclude that the blockchain technology adopted by cryptocurrencies nowadays has potential risks regarding to its forkable nature. This also influence the usage of current cryptocurrency wallet - there is no guarantee that the proposed transaction (or retrieved balance) from wallet is securely handled by the server if it contacts only one single machine.

Hence, the motivation of this research comes from this gap. The aim of this research is to develop a wallet making both transaction proposing and balance retrieving reliable and correct. During the first part of this research, a secure wallet will be developed, perfoming in a Red Belly Blockchain way

to avoid possible attacks targeting at fork chains. The Zero Knowledge Contingent Proof will try to add another guarantee so that either party within a transaction cannot cheat for its own benefits.

4 Methodologies and Proposed Contributions

4.1 Previous Work

4.1.1 Red Belly Blockchain

This research is on the basis of Red Belly Blockchain which is an unforkable blockchain running *Democratic Byzantine Fault Tolerance*. The development of secure wallet is build upon the implementation of Red Belly Blockchain.

In Red Belly Blockchain, all nodes are communicated through TCP+SSL. Nodes in this blockchain consists of nodes running Byzantine Consensus Algorithms - the *participants*, and external nodes communicating with the *participants*. A transaction or balance of an account can only be considered correct if more than one thirds of the *participants* agreed as required by DBFT.

4.2 Proposed Contributions

To develop a secure wallet, a secure protocol between server and clients is required. In this project, TCP and SSL is chosen to be the techniques for communications. I intent to implement *remote procedure call* for Red Belly Blockchain nodes first so that further functions can be achieved on client side.

Regarding to the client, I intend to implement secure wallet basic balance and transaction request functions complying with the same DBFT algorithm as Red Belly Blockchain nodes. In this sense, users of wallet can make sure that the balance and transaction are correctly obtained or proposed respectively.

In addition to server enhancement and secure wallet development, the pursuit of security also motivated this research to develop a Zero Knowledge Contingent Payments model for RBBC so that no party can cheats within such a 'contract'. The ZKCP implementation in RBBC is not like the one in Bitcoin who has scripts supporting this function while RBBC doesn't have.

4.3 Methodologies

The ultimate goal of this research is to develop a wallet that enables secure balance retrieving, transaction proposing, as well as trading. This would require design and development in both server side and client side.

The first part of this project is to add secure *remote procedure call* to Red Belly Blockchain. The *RPC* protocol should be through TCP and SSL. Once the *RPC* server is ready, Java and Android client could be developed based on it. After the client is developed, benchmarks on speed of balance retrieving and transaction proposing is required. And at the end of the above, I'll explore to develop a model to integrate *Zero Knowledge Contingent Payments* into the application.

5 Secure Wallet

In this report, secure wallet refers to two parts: the Red Belly Blockchain wallet and the Zero Knowledge Contingent Payments Model. This paper will explain the design and implementation of secure wallet in detail respectively.

5.1 Red Belly Blockchain Wallet

5.1.1 Preliminaries

Definitions

UTXO - the term **UTXO** stands for *Unspent Transaction Outputs*. UTXOs in Red Belly Blockchain act as the same role of it in Bitcoin, which records where spendable coins are in blockchain. To store UTXOs in Red Belly Blockchain, a table structure called UTXO table is used.

1. *UtxoOutput* In Red Belly Blockchain, one *unspent transaction output* consists of four fields: *spent* marks whether this output is consumed, *value* shows the amount of coin contained in this output, *address* showing where this output is sent to, and *script* shows the payee's public key, in other word, where this output comes from.

2. *UTXO Table* The UTXO table is the way Red Belly Blockchain nodes store UTXOs. As figure3 shows, the structure to store UTXOs includes two parts: a map structure mapping hash of transaction to list

Item	Description
Table	mapping between `<tx_hash : []Outputs>`
UserTxTable	mapping between `<user_address : [] hashes>`

Figure 3: UTXO Table Structure

of outputs and another map mapping account address to its related transaction's hash.

UTXOs in Red Belly Blockchain will be serialized into bytes during communication.

Transaction - transaction in Red Belly Blockchain is similar with transaction in Bitcoin. To define a transaction in Red Belly Blockchain, we have to define two definitions - the *Transaction Input* and the *Transaction Output*.

1. *Transaction Input* stands for the users' 'balance' for transaction. It consists of hash of transaction to be redeemed, index of its entry in UTXO table, and script standing for the account address that owns this UTXO.

2. *Transaction Output* serves as the transfer of coin. It contains the number of coins, the receiver's account address, and the payer's public key as its script. The output will give the money to the address specified. Meanwhile, an entry in UTXO table will be created to this output which makes sure that these coins can be used for another transaction.

In addition to lists of *Transcation Input*s and *Transaction Output*s, the address that the transaction is from as well the hash of the other parts of transaction is included.

Description of Red Belly Blockchain Transaction Syntax

First of all, users in Red Belly Blockchain is identified by their public keys pk_1 generated from ECDSA signature scheme. The address of one specific account is derived from public key using *Hash160*. In the simplest case, transaction T sends some number of coins from an address $address_1$

to another address $address_2$. The amount of coin is the *value* in *transaction*. The transaction T must have a pointer to a previous transaction T' such that T is valid only if T' is not redeemed before and has enough balance. Hence, in the simplest model, a transaction contains a following tuple $[T] := (TransactionInput : T', TransactOutput : T'', FromAddress : address_1, Hash : h([T]exceptforhash))$. To make a transaction valid, the same private key sk_1 corresponding to pk_1 should be used to sign it.

A transaction may also have multiple inputs and outputs which redeems a combination of several previous outputs and produce new outputs to multiple addresses. In Bitcoin network, the total value of one transaction's inputs can be larger than it outputs where the difference is called 'transaction fee' for rewarding miners. But we don't have such mechanism in Red Belly Blockchain, we simply conclude that there is no transaction fee or the fee equals to zero in our case.

Another kind of transaction is called *multisig* transaction. We will not consider this in the implementation of basic secure wallet. The concept and usage of it will be introduced in the ZKCP part of this paper.

5.1.2 Functions of Red Belly Blockchain Wallet

Just as its name suggests, the wallet application should have a basic function as a real wallet, namely to put money in, get the money and transfer it for goods. While this research is developing a digital one, the following functions is required to be included in the application:

1. **Configurations** - The wallet should be able to be configured according to various usage circumstances. Basically, the node addresses or DNS server addresses that the application communicates with should be able to configured within the application. In addition to this, the security level of application is also set to be configurable. In detail, the number of identical responses to accept as the correct response and the choice whether using an encryption during communication. The aim of the later one is to give users the right to choose from security or speed because possible latency may occur if there are more server to communicate or more complex encryption to compute.

2. **Account Settings** - As mentioned in section 5.1.1, an account is represented by its public key or derived addresses. The public key and

secret key pair is generated from ECDSA signature scheme. The application should be able to create new key pair or import existing key pair (indeed only private key needed to reconstruct).

3. **Retrieve Balance** - Balance is how many coins are available in UTXOs associated to this account. To get the balance of one specific account, we have two choice to implement, one is to compute the balance on server and return it to client, the other is to return UTXOs to client and let client traverse UTXOs itself. Here the second option is chosen considering that local UTXOs are also needed to propose transaction which gives enough reason to carry this time-consuming procedure. As for security concern, the UTXOs retrieved from Red Belly Blockchain nodes can only be considered correct if adequate number of identical responses - more than one thirds of the *participants* - had been received. Once the client get adequate confirmation of correct UTXOs, it will traverse the UTXOs to get its balance from unspent outputs.

4. **Propose Transaction** - The process to propose transaction has one more step compared to the process of balance retrieving. The prerequisite of make a transaction is to get correct UTXOs. Hence, the client will need to get adequate number of identical UTXOs first. After that, the client will existing outputs as the inputs of the newly created transaction, and create corresponding transaction outputs. After the above work is done, the client will wrap transaction inputs, transaction outputs, its own address and the hash of these fields together. The client then will sign the aggregated transaction and propose it to the *participants*. Once more than one thirds of nodes confirmed, the transaction could be regarded as accepted.

   Certainly there should be a way to give the target address for a transaction in the wallet. In our implementation, we use *Base64* encoded string of address bytes to represent an address. The application should be able to decode the address string. Despite of this, a more convenient way through *QR code* is provided. The wallet will be able to display QR code of its account address and scan others' QR code to decode addresses.

5. **Secure Storage** - The way to store information and data of secure wallet is another security concern. Though the consensus system Red

Belly Blockchain runs can tolerate malicious node, it's still dangerous to users' privacy. Hence, the application will not store any data into any format of files. Instead, we use the encrypted storage of Android OS. In this way, a system level security storage can be guaranteed.

5.1.3 Compatibility and External Libraries of Secure Wallet

To maximum the reusability and reduce the development cost, several existing libraries can be used in the implementation of Red Belly Blockchain secure wallet.

BitcoinJ - BitcoinJ is a Java library working in Bitcoin protocol originally. Thanks to the similar design of Red Belly Blockchain and Bitcoin, our secure wallet reuse the ECDSA signature scheme's implementation in BitcoinJ. In addition, the BitcoinJ library provides plenty of encoding and decoding methods for easily make account information readable.

Bounty Castle - Bounty Castle is a lightweight cryptography library for Java and C. Because of the lack of reconstruction from private key and customizable signature methods in BitcoinJ, this project introduce Bounty Castle for key pair generation and message signature.

ZXing - ZXing is a famous open-source library for multi-format 1D/2D barcode image processing written in Java. Secure wallet uses ZXing Android library to produce QR code, scan and decode it. Most recent Android models should be compatible with this library.

GSON - GSON is a Java library used to convert Java Objects into JSON representation and vice versa. The format of RPC calls between Red Belly Blockchain server and client is in JSON array string. We choose GSON for best conversion performance and reliability.

5.1.4 Implementation of Red Belly Blockchain Wallet

During this research, all development are carried on on Android Studio 2.3.3 and Android 3.0.0. The wallet follows the pure material design proposed by Android at the maximum extend.

When user enters the application, the default interface would be wallet activity where the balance of wallet will be displayed to illustrate the most basic function of a wallet function. If there is no account and server addresses set previously, the user may configure settings first through the left-side drawer. As for account, user can either choose to import a existing private

key and let the application reconstruct the original pair or generate a new key pair randomly. By whichever means, the keys will be stored securely in local storage.

To make a transaction, users can click the account to enter the balance activity where there is history of transactions and a button entry to pay someone. User can choose to type target address string in or scan QR code. If the user wish to receive money, he can also display his address representation in QR code. One may need to refresh so that the updated UTXOs is securely received.

While the user interface goes as the description, most of network communications, encryption, and encoding/decoding works in separate threads to make sure that no data modification or congestion will occur during the use of wallet. A simple Red Belly Blockchain server-client system model can been seen in Figure4. As the figure shows, clients communicate with multiple *participants* in Red Belly Blockchain in order to get adequate number of identical confirmations. All these communications are issued as an AsyncTask in Android which will not block or affect any operations in UI thread. The same AsyncTask is performed under both wallet activity and balance activity to send a request to get UTXOs for account of the wallet through RPC protocol. If enough number of identical UTXOs are received, the application will traverse them to calculate balance or aggregate usable outputs to propose a transaction. The same procedure is carried on for transaction while the client is only waiting for an acceptance confirmation rather than UTXOs, which is another RPC function definitely.

In this case, we can make sure that our wallet works securely even when there exist no more than one thirds of malicious participants in the network according to Red Belly Blockchain's DBFT algorithm. Also, thread safety and storage security in our implementation prevent possible violation during application execution.

5.1.5 Benchmark on RBBC Wallet

Four servers on Amazon Web Service in Oregon are set up for testing purpose. Each of them runs a dummy server. The time elapsed for retrieving balance and proposing transaction is recorded to demonstrate the speed performance of the RBBC secure wallet. The number of server client talks with is controlled by the security level within application settings. All other settings are set as default. And all transcations are between two pre-configured

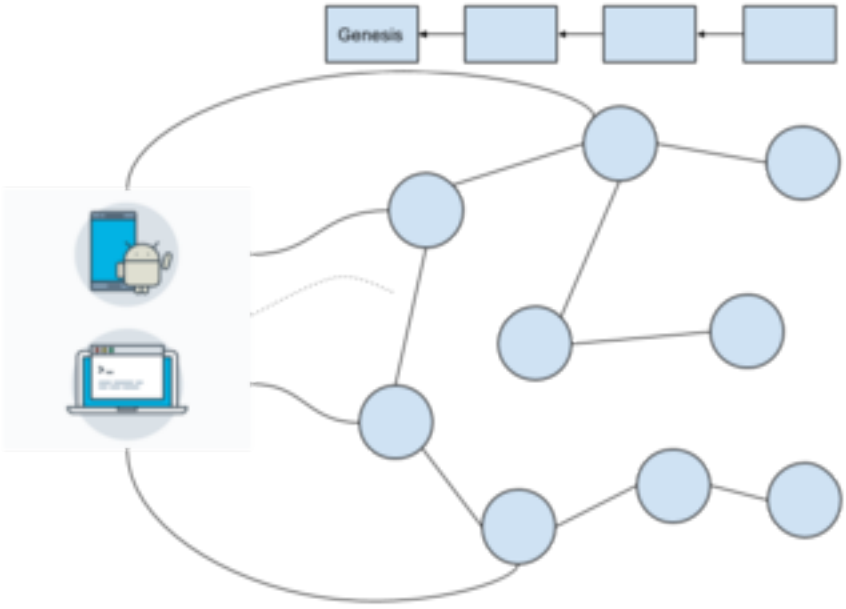

Figure 4: RBBC Server-Client System Model

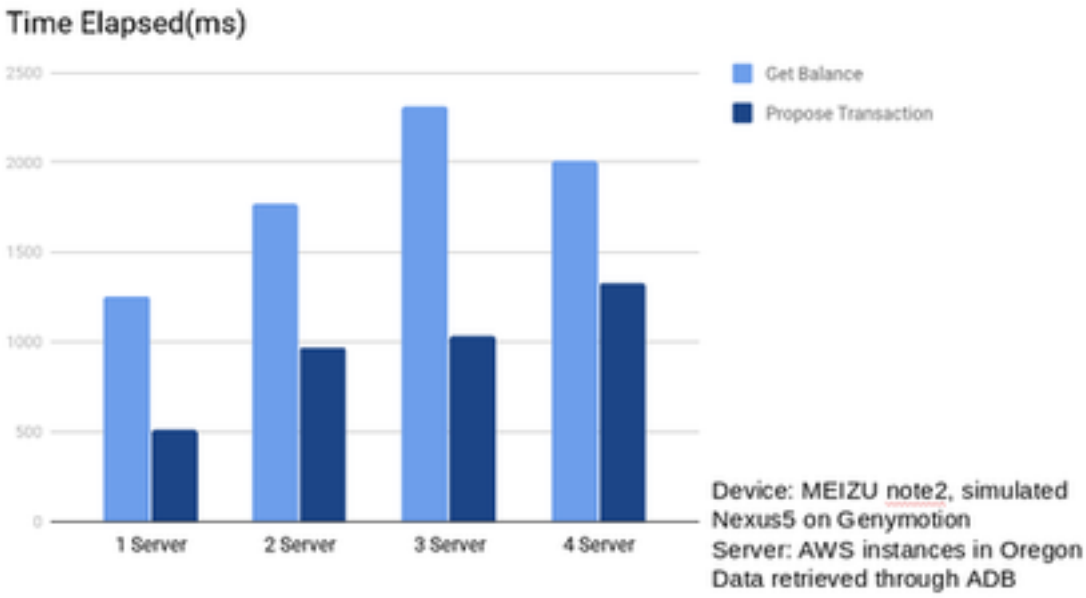

Figure 5: RBBC Secure Wallet Benchmark on Speed

account which is created along with genesis block. The result can be seen in Figure5.

5.2 Zero Knowledge Contingent Payments

The first Zero Knowledge Contingent Payments of Bitcoin is successfully made in 2016. The initial motivation of this exploration is to proove the robustness and capability of Bitcoin Scripts. Things are different in Red Belly Blockchain who has no scripts. Hence, this paper aims to find a proper model to fit *Zero Knowledge Contingent Payments* into Red Belly Blockchain, as well as the implementation of secure wallet.

5.2.1 Preliminaries

Definitions

Secret-Shared ECDSA Secret Key
Supposing there are two parties, namely buyer and seller, under our scene-

rio, they fix an elliptic curve over $(\mathbb{G}, O, g, +)$ over a field $\mathbb{Z}_p$. According to scheme of ECDSA, the secret key is a private integer $d \in \mathbb{Z}_{|G|}$. In our case, both buyer and seller input a security parameter and they both know the public key $pk = d \cdot g$ where d is the secret. Besides, the Seller and Buyer know $d_B, d_S \in \mathbb{Z}_{|G|}$ where $d_S \cdot d_B = d(mod|\ G\ |)$. [6] [9] The share protocol uses a commitment scheme in which eller make a commit $Commit(D_S)$ in exchange for buyer's secret key. If buyer's key is correct, seller will open his commitment afterwards.

Paillier Cryptosystem

Paillier cryptosystem is a probabilistic asymmetric algorithm for public key cryptography. The notable feature of Paillier cryptosystem is its homomorphic property along with its non-deterministic encryption. In this sense, giving only public key pk and the encryption of messages m_1 and m_2, one can compute the encryption of $m_1 + m_2$[18]. This scheme will be used in Section 5.2.3

Time-Locked Commitment

Zero Knowledge Contingent Payments in Red Belly Blockchain relies on Time-Locked Commitment schemes. To describe informally, the *Time-Locked Commitment* is no more than a standard $(Commit, Open)$ protocol except that it introduce a *force openning* mechanism where the receiver can open the commitment consuming a significant computational time. Though the time to *force open* a commitment differs from receivers, especially when taking mobile devices into accounts, we assume that this difference limited due to real-life parallelized efficiency.

Cut-and-Choose

Cut-and Choose is a common technique used in zero knowledge proof. The barebone idea of this is to perform multiple times of independent executions of a protocol. Statistically, if the seller can show parts of his answers are correct by uncovering those commitments, there is a high significant probability that the remaining ones are correct as well[12].

5.2.2 Secret-Shared Key Generation Protocol

In the first step of our ZKCP model, buyer and seller establish a connection between each other using the Secret-Share ECDSA scheme described in Section 5.2.1.

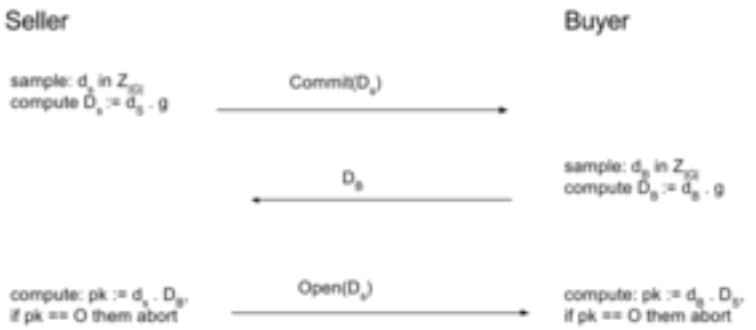

Figure 6: secret-shared key generation

The basic flow works like Figure 6.

5.2.3 Unique Signature Generation Protocol

The Unique Signature Generation Protocol uses protocol that generates secret-shared key stated in above Section 5.2.2. Before the execution of Unique Signature Generation Protocol, we assumes that two parties run the *secret-shared key generation protocol a* times.

Using the results from previous protocol, buyer and seller could sign a message m using the private key. At the first step of Unique Signature Generation Protocol, they jointly create a signing randomness R. Then the seller creates a new key using Paillier cryptosystem and sends the encryted d_S (the secret share of seller regarding to secret key d) to buyer. Buyer will continue to calculate the remaining part of the encryption of the unfinished signature and return the result to seller. At this stage, the seller is able to decrupts it and finally complete the signature σ. To make sure the trade remains unharmful if the seller is malicious, same commitment technique is applied here as well. Seller need to make a commitment $Commit(\sigma)$ of his signature σ and a time-locked commitment of his secret-shared secret key d_S so that buyer can *force open* the commitment if the time of contract expired.

5.2.4 SellWitness Protocol

To construct the final protocol to sell a witness as its name suggests, buyer need to create two transaction - T_1 and T_2. In our case, T_1 contains the money that buyer needs to pay and this will output to the address of the secret-shared key address or to a multisig escrow 't out of 2t-1' using the generated publick key t times and his own public key $t-1$ times. However, buyer will not reveal this until he is convinced by the seller. T_2 is a transaction redeeming T_1 to seller because the hash of T_1 can be known without really broadcasting it). The content of transaction T_2, named as m, is the message that needs to be signed later.

To simplify the description, this paper assumes $t = 1$ here. At current stage, T_2 is commonly known to both parties but seller needs T_1 to spend T_2 while T_1 is private to buyer. Meanwhile buyer need learns the signature σ of T_2 so that he can know the secret key that seller committed. Or after some time, the buyer may *force open* seller's $Commit(d_S)$.

Here the *cut-and-choose* will be used again. Supposing there exist a zero knowledge proof of knowledge F such that it has a function to extract witness satisfying $Extract_F(Buyer, Challenge_1, Answer_1, Challenge_2, Answer_2) = x$ and $f(x) = true$ if only anwsers are all correct and challenges are not the same.

Buyer will create multiple challenge pairs, $(c_1, c_2)_i$ respectively. And Seller calculates responses to each of them and commit the result encryption r_1 and r_2 using his asymmetric cyphers (the same secret keys used to sign T_2). Then Buyer selects certain number of challenge pairs and challenge Seller using a random one within each pair. The seller opens the corresponding commitments of secret key in return. Once buyer verifies all these commitments without error. The seller will open all commitments of encrypted answers to Buyer's challenges. To be noticed, the commitment of secret key of seller is not revealed yet.

Now the buyer will broadcast the private transaction T_1. The Seller can only spend it by revealing signature of T_2 so that he can propose T_2 to be accepted. In this stage, buyer can finally derive the secret key created during the construction of secret-shared key pair. Using this key, buyer then can use any pair of responses to decrypt the correct answer and work out the witness as the return of this trade.

6 Discussion

This research receives constructive results from the development of secure wallet and several inspired thoughts from the scratch of the zero knowledge contingent payments model in Red Belly Blockchain. This section will have a discussion on both side in the following contents.

6.1 Red Belly Blockchain Secure Wallet

To evaluate the results gained from secure wallet, it's more proper to have a look into its **performance** and **security** aspects separately.

6.1.1 Performance of Secure Wallet

Definitely, the number of test samples is not adequate right now due to time issues. We can still conclude that the speed of transcation which indeed the time to derive a commonly agreed UTXOs is satasfying considering that the client is communicating with multiple remote servers across ocean. This proves the feasibility to deploy Red Belly Blockchain into real life applications, not limited to apllication like wallet.

To be noticed, **scalability** is another advantage shown in our results. Due to the 'independent threads' implementation of secure wallet, the number of *participants* we are communicating with has few influence on the time elapsed to retrieve a correct response.

However, one potential threats that is not taken into consideration in our case is the size of UTXOs. The number clients limits the size growth in our network which leads to uncertain results if we have a large and busy blockchain network. Also the time needed to traverse a local UTXOs storage may also increase as well. The performance of SQLite in Android is not measured as well.

6.1.2 Security of Secure Wallet

As for security concern, this research mainly concentrates on communications and local storage. The benefits and tradeoffs of communicating with multiple *participants* are both obvious. As long as the wallet follows principles, UTXOs obtained can be regarded as correct. Potential risks comes from Android local storage. The implementation of secure wallet avoids to leave

possibility for any modification in local storage, but there is no guarantee that permission is well managed within the phone. Possible modification or injection still exists on OS level.

6.2 Zero Knowledge Contingent Payments in Red Belly Blockchain

The Zero Knowledge Contingent Payments model in Red Belly Blockchain is rather a model or future work to go. The implementation of it is based on common safety assumptions of well-known techniques mentioned in Section 5.2.1. Hence, the proof is well addressed under real life applications. In addition to this, because of the lack of scripts in Red Belly Blockchain, we have to adopt the *cut-and-choose* technique which requires a lot of redundancies. The time complexity of it can be simply co nclude as $O(tn)$ where we apply *cut-and-choose* twice.

7 Future Work

7.1 Secure Wallet

Obviously more tests involving more *participants* should be carried on in order to benchmark the performance and find out the bottleneck so that a better solution can be derived to make the wallet easy to use.

In addition to this, a measure on the performance when coming to large size of UTXOs is needed. Both the lag of network and the time to traverse the whole UTXOs are required to be analyzed. Tests on possible defects on storage can be carried as well.

7.1.1 ZKCP model of RBBC

As the discussion in Section 6.2, the ZKCP model requires further proof as well as improvements in efficiency. Also a timing mechanism is n eeded if we will integrate the ZKCP model into Red Belly Blockchain as it uses the *time-locked commitment*.

8 Conclusion

In conclusion, the risks behind current cryptocurrencies technology motivates this research to explore a way to ensure security in both blockchain levels and contracts levels, referring to the development of secure wallet and the design of zero knowledge contingent payments respectively in our case. By the help of Red Belly Blockchain with Democratic Byzantine Consensus System running, the first version of secure wallet provides a satisfying performance under basic tests. The ZKCP model also shows a potential to secure transaction with less support from system-level comapred to Bitcoin Scripts or Ethereum's Smart Contracts. Although there is no build or benchmarks at current stage, the capability to involve a mechanism ensuring no parties can cheat within a transaction is of vital importance considering the security issue that we emphasize most in this research.

The application or future of this research can goes into a wider range not limited to a secure wallet on Android or a draft model of the *zero knowledge contingent payments*. Further research can be conducted in order to find out the better usage or improvements for a secure blockchain as our pursuit.

References

[1] Wacław Banasik, Stefan Dziembowski, and Daniel Malinowski. *Efficient Zero-Knowledge Contingent Payments in Cryptocurrencies Without Scripts*, pages 261–280. Springer International Publishing, Cham, 2016.

[2] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 287–304. IEEE, 2015.

[3] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services.

[4] Ioannis Chatzigiannakis, Apostolos Pyrgelis, Paul G. Spirakis, and Yannis C. Stamatiou. Elliptic curve based zero knowledge proofs and their applicability on resource constrained devices. 2011.

[5] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. (leader/randomization/signature)-free byzantine consensus for consortium blockchains. *CoRR*, abs/1702.03068, 2017.

[6] Steven Goldfeder, Rosario Gennaro, Harry Kalodner, Joseph Bonneau, Joshua A Kroll, Edward W Felten, and Arvind Narayanan. Securing bitcoin wallets via a new dsa/ecdsa threshold signature scheme, 2015.

[7] Vincent Gramoli. On the danger of private blockchains. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL'16)*, 2016.

[8] Vincent Gramoli. From blockchain consensus back to byzantine consensus. *Future Generation Computer Systems*, 2017.

[9] Amir Herzberg, Stanislaw M Jarecki, Hugo M Krawczyk, and Marcel M Yung. Method and system for a public key cryptosystem having proactive, robust, and recoverable distributed threshold secret sharing, April 29 1997. US Patent 5,625,692.

[10] Yael Tauman Kalai and Ran Raz. Succinct non-interactive zero-knowledge proofs with preprocessing for logsnp. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 355–366. IEEE, 2006.

[11] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[12] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.

[13] G Maxwell. Zero knowledge contingent payment. 2011. *URl: https://en. bitcoin. it/wiki/Zero_Knowledge_Contingent_Payment (visited on 05/01/2016)*.

[14] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with t¡ n/3, o (n2) messages, and o (1) expected time. *Journal of the ACM (JACM)*, 62(4):31, 2015.

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

[16] Christopher Natoli and Vincent Gramoli. The blockchain anomaly. In *Proceedings of the 15th IEEE International Symposium on Network Computing and Applications (NCA'16)*, pages 310–317. IEEE, Oct 2016.

[17] Christopher Natoli and Vincent Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *Proceedings of the 47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'17)*. IEEE, Jun 2017.

[18] Pascal Paillier et al. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, volume 99, pages 223–238. Springer, 1999.

[19] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.

[20] Bitcoin Wiki. Zero knowledge contingent payment, 2011.

[21] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151, 2014.

[22] Fucai Zhou, Yuxi Li, Qingshi Zhou, Jingwei Miao, and Jian Xu. The electronic cash system based on non-interactive zero-knowledge proofs. *International Journal of Computer Mathematics*, 93(2):239–257, 2016.