Design of Anonymous Endorsement System in Hyperledger Fabric

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Subhra Mazumdar

[Roll No: CS1609]

under the guidance of

Dr. Sushmita Ruj
Assistant Professor
Cryptology and Security Research Unit

Indian Statistical Institute
Kolkata-700108, India

July 2018

To my family, friends and Hyperledger community

CERTIFICATE

This is to certify that the dissertation entitled **"Design of Anonymous Endorsement System in Hyperledger Fabric"** submitted by **Subhra Mazumdar** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Sushmita Ruj
Assistant Professor,
Cryptology and Security Research Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgments

I would first like to express my sincere gratitude to my advisor, *Dr. Sushmita Ruj*, Cryptology and Security Research Unit, Indian Statistical Institute, Kolkata, for her continuous support, advice, encouragement and motivation. Her guidance and knowledge helped me in pursuing good research and writing of this thesis. It is a privilege to have a supervisor who has constantly supported me, both academically and personally.

I would also like to thank *Prabal Banerjee*, Cryptology and Security Research Unit, Indian Statistical Institute, Kolkata, who provided insight and valuable suggestions that greatly assisted the research.

I would like to express my gratitude to all the community members of Hyperledger project, for the discussions and prompt reply to all my queries.

I take this opportunity to sincerely acknowledge the contributions of all the teachers of Indian Statistical Institute, who have deeply influenced my research acumen.

I thank my parents, for being my pillar of strength and supporting me in all my endeavours. Last but not the least, I want to thank my friends, for keeping me motivated at times when I felt like giving up.

Abstract

Permissioned Blockchain has become quite popular with enterprises forming consortium since it prioritizes trust over privacy. One of the popular platforms for distributed ledger solution, *Hyperledger Fabric*, requires a transaction to be *endorsed* or approved by a group known as endorsers as per the specifications in the endorsement policy. To endorse a transaction, an endorser mentions its identity along with the signature so that it can be verified. However, for certain transactions, difference in opinion may exist among endorsers. Disclosure of identity of endorser may lead to conflict within the consortium. In such cases, an endorsement policy which allows an endorser to support a transaction discreetly but simultaneously takes into account the decision of the majority is preferred. As a solution, we propose an Anonymous Endorsement System which uses a threshold endorsement policy. For hiding the identity of endorsers, a new ring signature scheme, called *Fabric's Constant-Sized Linkable Ring Signature* (FCsLRS) with *Transaction-Oriented* linkability has been proposed. We have implemented the signature scheme in Golang and analyzed its security and performance by varying the RSA modulus size. Feasibility of implementation is supported by experimental analysis. Signature generation and verification is quite fast, with execution time remaining constant irrespective of change in message length or endorsement set size for a given RSA modulus value. Lastly, we also discuss the integration of the scheme with existing version of Hyperledger Fabric.

Keywords: *Permissioned Blockchain, Hyperledger Fabric, Anonymous Endorsement System, Fabric's Constant-Sized Linkable Ring Signature (FCsLRS), Transaction-Oriented linkability.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the explosion in the magnitude of transaction carried out worldwide, complexity and cost of functioning has also magnified. Further, presence of third party for validation of these transactions leads to duplication of effort as well as issue of trust. With the onset of net banking and online purchases, users are more vulnerable to fraud, cyberattacks. The identities of participants in a network remains under constant threat of getting exposed, if the controller of network gets compromised.

Blockchain, the leading technology underpinning bitcoin and other cryptocurrencies, offers a solution to all such problems. It has the potential to revolutionize the financial services by replacing intermediaries involved in any business transaction, thereby reducing risks in the event of breach of trust. This leads to transparency and accountability for services offered by banking, insurance etc. It allows people to create digital property without depending on third parties.

1.1 Blockchain - Decentralized trust

In technical terms, blockchain is a shared, distributed ledger that facilitates the process of recording transactions and tracking *assets* in a business network [28]. An *asset* can be tangible - a house, a car, cash, land - or intangible like intellectual property, such as patents, copyrights, or branding. Each block contains a unique identifier, timestamped batches of recent valid transactions, and the hash of the previous block. This results in blockchain being tamper-proof and immutable where addition of subsequent block strengthens the verification of the previous block and hence the entire blockchain. The blockchain architecture supports peer-to-peer replication whereby participants(node) in the network update the shared ledger on the occurrence of a transaction. Each node can receive or send transactions to other nodes, and the data is synchronized across the network as it is transferred.

Some Applications:

- **Bitcoin**: One of the very popular use case for blockchain is *Bitcoin*. It acts as a shared ledger for the cryptocurrency, recording transactions. Bitcoin is not governed by a central monetary authority. It is cost effective as well as secure. Other relevant use cases are
- **Financial Services**: In cross border remittance, commercial financing of its business partners by a company.
- **Insurance**: It eliminates the cost of processing insurance claims, reduces the instance of insurance fraud and improves customer satisfaction.
- **Supply Chain Management**: With no single authority "owns" the provenance information, it lead to reductions in time taken to diagnose and remedy a fault improving system utilization.
- **Healthcare**: Ensures secure storage of medical data, retrieval of patient's medical history is easier and medical claim processing time reduces.
- **Internet of Things(IoT)**: As machines interact with one another, any relevant interactions can be reported by the machines and recorded in the blockchain to increase efficiency and accuracy, automating IoT processes.
- **Other applications**: Retail, energy management, voting, real estate etc.

Types of Blockchain

There are two types of blockchain [24],[14]:
Permissionless Blockchains: It is open and decentralized where any node can join and leave the network at any point of time. There is no central entity which manages the membership, or which could ban illegitimate members. Each party can choose to run a node for the blockchain and participate in transaction verifications (via the mining mechanism), as well as create smart contracts on the network. These networks employ a crypto-economic model (driven by *proof-of-work*, *proof-of-stake* consensus algorithm) that incentivizes more participants to join the network. Use of cryptocurrency is necessary to simplify exchange of value among participants as well as rewarding them for their contributions.
Example: Bitcoin, Ethereum.

Permissioned Blockchains: It is a closed ecosystem in which each node needs permission to participate in the network, suitable for business applications. Such nodes are involved in validation of transaction and execution of smart contracts. This type of blockchain is suitable for an organization or consortium of organization to efficiently exchange information and record transactions, retaining confidentiality

and privacy unlike permissionless blokchains. No monetary tokens or incentivization mechanism involved due to the nature of these business networks. Computationally inexpensive consensus mechanisms like *Practical Byzantine Fault Tolerance (PBFT), Simplified Byzantine Fault Tolerance (SBFT)* are used which results in substantially better scalability and performance.
Example: Kadena, Tendermint, Hyperledger Fabric.

1.2 Problem Statement & Motivation

In permissionless blockchains, the miners and other participants of the network can stay anonymous. No central authority has total controls over the functioning of the system. But for political or business organizations, transparency is of utmost importance for fair governance. On the contrary, the permissioned counterpart is governed by the members of the blockchain business network. Economic incentives, code quality, code changes, and power allocation among peers are based on the business dynamics for which the network has been designed and built.

One of the most popular open-source permissioned blockchain framework, *Hyperledger Fabric*, is one of the most mature and active project of Hyperledger. It is quite scalable and robust, hence used mainly for enterprise purpose. It's architecture is illustrated via a diagram in Fig. 1.1. A membership service provider (MSP) is responsible for maintaining the identities of all nodes in the system - *clients*,*orderer* and *validator or peer*. It issues credentials in the form of cryptographic certificates which is used for the purpose of authentication and authorization.

Let us the define the role played by each of the entities given in Fig. 1.1 [23].

- **Client**: Represented as *Application* in the diagram, the client represents the entity that acts on behalf of an end-user. It must connect to a peer(of its choice or pre-defined) for communicating with the blockchain. Clients are responsible for invoking transactions.

- **Peer**: Peers are a fundamental element of the network because they host ledgers and smart contracts. A peer receives ordered state updates in the form of blocks from the ordering service and maintain the state and the ledger. It can additionally take up a special role of an *endorsing peer*, or an *endorser*. The special function of an endorsing peer occurs with respect to a particular chaincode and consists in endorsing a transaction based on specified endorsement policy, before it is committed.

- **Ordering Service Nodes or Orderers**: The ordering service establishes the total order of all transactions in Fabric by reaching a consensus among all peer nodes. Blocks are disseminated to all the peer nodes via peer-to-peer gossip service. Orderers are entirely unaware of the application state, and do not participate in the execution nor in the validation of transactions. An ordering

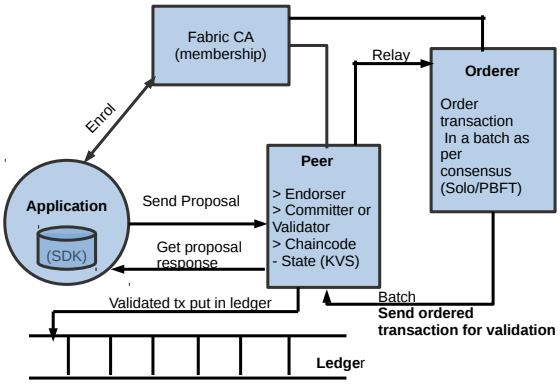


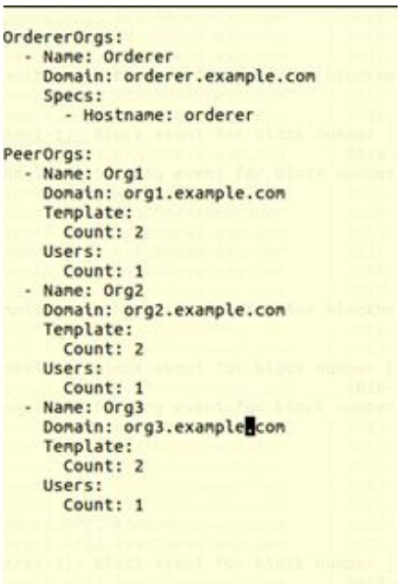
Figure 1.1: Hyperledger Fabric Architecture

service atomically broadcasts state updates to peers and establishes consensus on the order of transactions.

- **Fabric CA(Certification Authority)**: It is a private root CA provider capable of managing digital identities of Fabric participants that have the form of X.509 certificates. Because Fabric-CA is a custom CA targetting the Root CA needs of Fabric, it is inherently not capable of providing SSL certificates for general/automatic use in browsers.

Any interaction among nodes occur through messages that are authenticated via digital signatures. A client, willing to put the transaction on the ledger, signs and submits a transaction proposal to the set of endorsers, specified by underlying endorsement policy, for execution (for example as per the configuration defined in Fig. 1.2). It is deemed as valid if and only if it collects sufficient number of endorsement for satisfying the endorsement policy. Endorsing peer after verifying the correctness of the transaction executes it against the current state database. Signature of the endorser along with endorser id is attached to *response* (as in Fig. 1.3). Peer nodes, responsible for validation, verifies it by inspecting the identity and signature of each endorser. This ensures transparency in transaction flow (explained in details in Section 2.1.3) where every details can be monitored.

However, transparency does not always ensure fairness in governance. Consider a situation where a cluster of members or organization more powerful than the rest of the members of consortium. They are the ones usually deciding the rules and policies, which is biased in their favour. Rest have very little say in such matter and might lend support by default, for fear of adverse consequences. Even in an egalitarian-styled consortium, openly voicing one's opinion may cause internal conflict. There exists possibility of external conflict as well, where particular organizations gets sabotaged for making decision against public interest. In such cases where none of the parties can be trusted with sensitive information, privacy must be given preference over transparency.



```
OrdererOrgs:
  - Name: Orderer
    Domain: orderer.example.com
    Specs:
      - Hostname: orderer

PeerOrgs:
  - Name: Org1
    Domain: org1.example.com
    Template:
      Count: 2
    Users:
      Count: 1
  - Name: Org2
    Domain: org2.example.com
    Template:
      Count: 2
    Users:
      Count: 1
  - Name: Org3
    Domain: org3.example.com
    Template:
      Count: 2
    Users:
      Count: 1
```

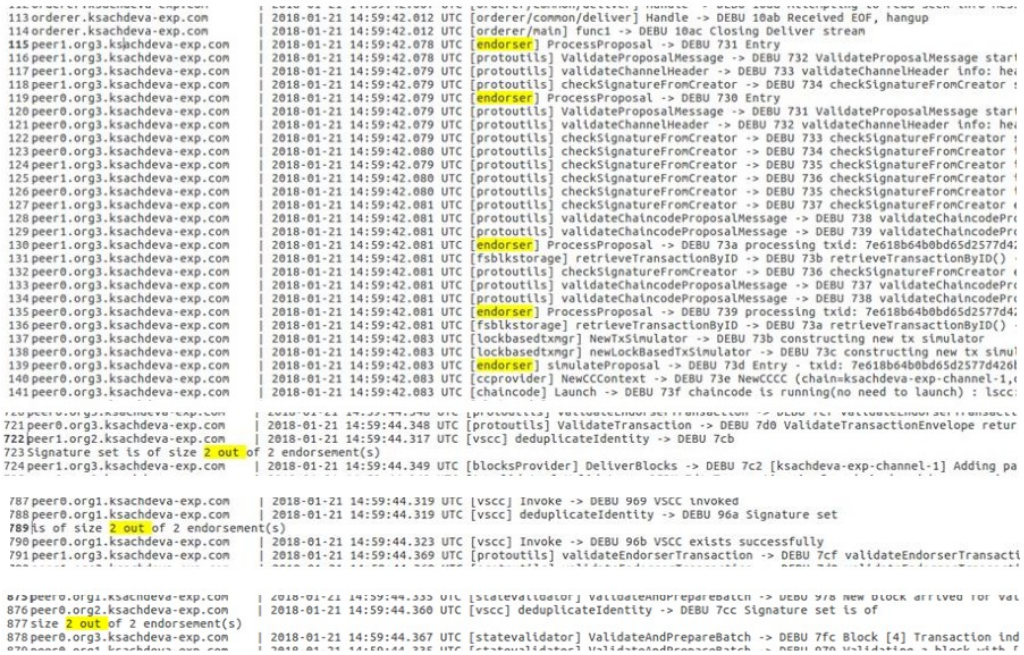
Consider a configuration :

- A consortium of 3 organization
- Each organization has 2 members : One admin and one regular member.
- **Peer nodes :** Ogr1admin , Org1 member, Ogr2 admin , Org2 member, Ogr3 admin , Org3 member.
- **Orderer node** : Orderer.

Endorsement Policy : 2 out of 2
[AND(Org1.admin, Org1.member)]
OR [AND(Org2.admin, Org2.member)]
OR [AND(Org3.admin, Org3.member)]

Executed on Hyperledger Fabric v1.0.5

Figure 1.2: crypto-config.yaml template in Hyperledger Fabric



```
113 orderer.ksachdeva-exp.com      | 2018-01-21 14:59:42.012 UTC [orderer/common/deliver] Handle -> DEBU 10ab Received EOF, hangup
114 orderer.ksachdeva-exp.com      | 2018-01-21 14:59:42.012 UTC [orderer/main] func1 -> DEBU 10ac Closing Deliver stream
115 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.078 UTC [endorser] ProcessProposal -> DEBU 731 Entry
116 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.078 UTC [protoutils] ValidateProposalMessage -> DEBU 732 ValidateProposalMessage start
117 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.079 UTC [protoutils] validateChannelHeader -> DEBU 733 validateChannelHeader info: he
118 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.079 UTC [protoutils] checkSignatureFromCreator -> DEBU 734 checkSignatureFromCreator
119 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.079 UTC [endorser] ProcessProposal -> DEBU 730 Entry
120 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.079 UTC [protoutils] ValidateProposalMessage -> DEBU 731 ValidateProposalMessage start
121 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.079 UTC [protoutils] validateChannelHeader -> DEBU 732 validateChannelHeader info: he
122 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.079 UTC [protoutils] checkSignatureFromCreator -> DEBU 733 checkSignatureFromCreator
123 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.080 UTC [protoutils] checkSignatureFromCreator -> DEBU 734 checkSignatureFromCreator
124 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.079 UTC [protoutils] checkSignatureFromCreator -> DEBU 735 checkSignatureFromCreator
125 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.080 UTC [protoutils] checkSignatureFromCreator -> DEBU 736 checkSignatureFromCreator
126 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.080 UTC [protoutils] checkSignatureFromCreator -> DEBU 735 checkSignatureFromCreator
127 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.081 UTC [protoutils] checkSignatureFromCreator -> DEBU 737 checkSignatureFromCreator
128 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.081 UTC [protoutils] validateChaincodeProposalMessage -> DEBU 738 validateChaincodePro
129 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.081 UTC [protoutils] validateChaincodeProposalMessage -> DEBU 739 validateChaincodePro
130 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.081 UTC [endorser] ProcessProposal -> DEBU 73a processing txId: 7e618b64b0bd65d2577d4
131 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.081 UTC [fsblkstorage] retrieveTransactionByID -> DEBU 73b retrieveTransactionByID()
132 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.081 UTC [protoutils] checkSignatureFromCreator -> DEBU 736 checkSignatureFromCreator
133 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.081 UTC [protoutils] validateChaincodeProposalMessage -> DEBU 737 validateChaincodePro
134 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.081 UTC [protoutils] validateChaincodeProposalMessage -> DEBU 738 validateChaincodePro
135 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.081 UTC [endorser] ProcessProposal -> DEBU 739 processing txId: 7e618b64b0bd65d2577d4
136 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.081 UTC [fsblkstorage] retrieveTransactionByID -> DEBU 73a retrieveTransactionByID()
137 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.083 UTC [lockbasedtxmgr] NewTxSimulator -> DEBU 73b constructing new tx simulator
138 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.083 UTC [lockbasedtxmgr] newLockBasedTxSimulator -> DEBU 73c constructing new tx simu
139 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.083 UTC [endorser] simulateProposal -> DEBU 73d Entry - txid: 7e618b64b0bd65d2577d426
140 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.083 UTC [ccprovider] NewCCContext -> DEBU 73e NewCCCC (chain=ksachdeva-exp-channel-1,
141 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:42.083 UTC [chaincode] Launch -> DEBU 73f chaincode is running(no need to launch) : lscc

721 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:44.348 UTC [protoutils] ValidateTransaction -> DEBU 7d0 ValidateTransactionEnvelope retur
722 peer1.org2.ksachdeva-exp.com   | 2018-01-21 14:59:44.317 UTC [vscc] deduplicateIdentity -> DEBU 7cb
723 Signature set is of size 2 out of 2 endorsement(s)
724 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:44.349 UTC [blocksProvider] DeliverBlocks -> DEBU 7c2 [ksachdeva-exp-channel-1] Adding pa

787 peer0.org1.ksachdeva-exp.com   | 2018-01-21 14:59:44.319 UTC [vscc] Invoke -> DEBU 969 VSCC invoked
788 peer0.org1.ksachdeva-exp.com   | 2018-01-21 14:59:44.319 UTC [vscc] deduplicateIdentity -> DEBU 96a Signature set
789 is of size 2 out of 2 endorsement(s)
790 peer0.org1.ksachdeva-exp.com   | 2018-01-21 14:59:44.323 UTC [vscc] Invoke -> DEBU 96b VSCC exists successfully
791 peer1.org3.ksachdeva-exp.com   | 2018-01-21 14:59:44.369 UTC [protoutils] validateEndorserTransaction -> DEBU 7cf validateEndorserTransacti

876 peer0.org2.ksachdeva-exp.com   | 2018-01-21 14:59:44.360 UTC [vscc] deduplicateIdentity -> DEBU 7cc Signature set is of
877 size 2 out of 2 endorsement(s)
878 peer0.org3.ksachdeva-exp.com   | 2018-01-21 14:59:44.367 UTC [statevalidator] ValidateAndPrepareBatch -> DEBU 7fc Block [4] Transaction ind
```

Endorsers: Org3.admin, Org3.member. Validation done by all 6 peer nodes.

Figure 1.3: Tracking endorser's identity

Blockchain was created to solve the specific problem of providing trust when all participants are anonymous. In permissionless setting, anyone can join the network without credentials. This is not true in the permissioned setting, where a participant's identity is know by everyone within the network. However, recently Hyperledger Fabric intends to integrate *Identity Mixer MSP (Membership Service Provider)*[10] in it's future releases where client and/or peer nodes can sign the transaction by remaining anonymous, generating unlinkable signature. With the aid of zero-knowledge proofs, a peer node can prove the correctness of the generated signature, without the verifier obtaining extra information. A privacy preserving distributed ledger, `zkLedger` [31] claims to support strong transaction as well as participants privacy (for hiding transaction value it uses Pedersen commitments) but at the same time provides fast and provably correct auditing using Schnorr-type non-interactive zero-knowledge proofs. But none of them address the concern of hiding the identity of endorser. This motivated us to design of a bias-free *endorsement system*, allowing provision of an endorsement policy which takes into account substantial amount of support by members of a pre-specified endorsement set without explicitly specifying who all must endorse.

1.3 Designing a new Endorsement System

A threshold endorsement policy serves the purpose which requires atleast t out of $n, (1 \leq t \leq n/2)$ endorsers to approve of the transaction without explicitly mentioning their identity. Any implementation of threshold cryptosystem might seem an obvious answer, but it does not ensure perfect anonymity. A trusted third party is requested to generate the keys shares and distribute among signing parties (*Kate et al. [25]* proposed a scheme for distributed key generation scheme but it comes with fair amount of computation overhead). Any entity performing the task of signature combination in threshold signature scheme must know id value of the signer so that it can compute the Lagrange co-efficient. But our motive is to achieve a *t-out-of-n* endorsement without revealing the identity of endorsers to any member within the network, including the endorsers themselves. To alleviate this problem, we make each endorser sign the transaction anonymously using a very popular cryptographic protocol, *Ring Signature*. It allows any signer to create an ad-hoc group (termed as *ring*) and produce a signature on behalf of the group, without revealing the individual signer's identity. This offers "unconditional anonymity" not attainable through generic digital signature schemes, without requiring any complex setup procedure. Detachment of identity of signer from the endorsement leads to problem of double-signing whereby a member tries to endorse more than once. We prevent this by making multiple signatures generated by a particular signer for a particular session linkable. For accomplishment of threshold endorsement policy, verifiers resort to simple counting of each valid endorsement till it crosses the threshold value.

1.4 Our Contributions

In this thesis, we have made the following contributions.

- We have proposed an *Anonymous Endorsement System* which implements a simple threshold endorsement policy which requires at least t out of n endorsers to approve of the transaction without explicitly mentioning their identity.
- We have given the construction of a constant sized linkable ring signature scheme, Fabric's Constant-Sized Linkable Ring Signature (*FCsLRS*), to hide the identity of an endorser.
- A new linking criterion, called as *"transaction-oriented"* linkability, is used* which prevents an endorser from signing same transaction more than once.
- We have implemented this scheme in Golang (refer [2]) and analysed its performance.
- A detailed description of the integration of the scheme with existing framework of Hyperledger Fabric has been discussed.

1.5 Organization of the thesis

`Chapter 2` discusses about Hyperledger Fabric - it's components, endorsement policy and transaction flow. Few related works on various signature schemes and its application in blockchain system has been studied. Later, under preliminaries, we define the mathematical notations and basic building blocks needed for the new endorsement system. In `Chapter 3`, we describe our proposed anonymous endorsement system. Construction of Fabric's Constant-Sized Linkable Ring Signature (*FCsLRS*) scheme is stated in `Section 3.1`. In `Chapter 4`, we describe our definition of security model. `Chapter 5` gives the Performance Analysis. `Chapter 6` gives details of possible steps needed to integrate the *Fabric's Constant-sized linkable ring signature (FCsLRS) scheme with Fabric*. Conclusion and Future work is mentioned `Chapter 7`.

*Possibility of double signing may arise for a given transaction when more than one endorsement is required.

Chapter 2

Background

2.1 Hyperledger Fabric

Upto version 0.6, Fabric used to follow the order-execute architecture which had several limitations. From version 1.0 onwards, it has been revamped to *execute-order-validate*, ensuring resiliency, flexibility, scalability and confidentiality.

2.1.1 Hyperledger Fabric Components

Fabric is written in Go and uses the gRPC framework for communication between clients, peers, and orderers. The following definitions are as stated in[3].

Membership Service

The membership service provider (MSP) maintains the identities of all nodes in the system (clients, peers, and orderers) and is responsible for issuing node credentials that are used for authentication and authorization. Since Fabric is permissioned, all interactions among nodes occur through messages that are authenticated, typically with digital signatures. *Fabric CA* can also be used to generate the keys and certificates needed to configure an MSP.

Ledger

The ledger component at each peer maintains the ledger and the state on persistent storage and enables simulation, validation, and ledger-update phases. Broadly, it consists of a block store and a peer transaction manager(PTM). For transaction validation , the PTM validates all transactions in a block sequentially. This checks whether a transaction conflicts with any preceding transaction (within the block or earlier).

Chaincode Execution

Chaincode is executed within an environment loosely coupled to the rest of the peer, which supports plugins for adding new chain- code programming languages. Currently Go, Java, and Node are supported.

System Chaincodes

System Chaincode (SCC) is a chaincode built with the peer and run in the same process as the peer. SCC is responsible for broader configurations of fabric behavior, such as timing and naming services. They run as part of the peer process as opposed to user chaincodes that run in separate docker containers. As such they have more access to resources in the peer and can be used for implementing features that are difficult or impossible to be implemented through user chaincodes. Examples of System Chaincodes are *ESCC (Endorser System Chaincode)* for endorsing proposals, *QSCC (Query System Chaincode)* for ledger and other fabric related queries and *VSCC (Validation System Chaincode)* for validating a transaction at commit time respectively.

To invoke a transaction, in the current implementation, the client sends a `PROPOSE` message to a set of endorsing peers of its choice. The set of endorsing peers for a given chaincodeID is made available to client via peer(a client may connect to a submitting peer for invoking transactions and obtaining results), which in turn knows the set of endorsing peers from endorsement policy . That said, some endorsers could be offline, others may object and choose not to endorse the transaction. The submitting client tries to satisfy the policy expression with the endorsers available.

2.1.2 Endorsement policies

An *endorsement policy*, is a condition which states which endorser/s must endorse a transaction. Blockchain peers have a pre-specified set of endorsement policies, which are referenced by *"deploy transaction"* that installs specific chaincode. Peers can additionally take up a special role of an endorsing peer, or an endorser. A transaction is declared valid only if it has been endorsed according to the policy. An invoke transaction for a chaincode will first have to obtain an endorsement that satisfies the chaincode's policy or it will not be committed. The evaluation of an endorsement policy predicate must be deterministic. An endorsement shall be evaluated locally by every peer such that a peer does not need to interact with other peers, yet all correct peers evaluate the endorsement policy in the same way.

Examples of endorsement policy

The predicate may contain logical expressions and evaluates to `TRUE` or `FALSE`.

Let the endorsement set be $\mathcal{E} = \{A, B, C, D, E, F\}$. Some examples are:

- All 5 endorsers in $\mathcal{E}$ must approve of the transaction.
- Any one endorser in $\mathcal{E}$ must approve of the transaction.
- Endorser A "AND" endorser B approve of the transaction.
- Either endorser C "OR" endorser B approve of the transaction.
- Any 3 out of 5 endorsers approve of the transaction.
- Suppose there is an assignment of "stake" or "weights" to the endorsers, like $\{A = 3, B = 2, C = 2, D = 1, E = 1, F = 1\}$, where the total stake is 10: The policy requires valid endorsement from a set that has a majority of the stake (i.e., a group with combined stake strictly more than 5), such as $\{A, B, C\}$ or $\{B, C, D, E, F\}$ etc.

2.1.3 Transaction flow

The transaction flow is illustrated in Fig. 2.1. It involves three main phases - execution (step $\langle 1 \rangle$ & step $\langle 2 \rangle$), ordering (step $\langle 3 \rangle$) and validation (step $\langle 4 \rangle$)(as in [3]).

Execution Phase

In the execution phase, clients sign and send the transaction proposal to one or more endorsers for execution. A *"proposal"* contains the identity of the submitting client, the transaction payload in the form of an operation to execute, parameters, and the identifier of the chaincode, transaction id and other miscellaneous information. On receipt of the proposal by the endorsers, it simulate the proposal against it's local blockchain state , by executing the operation on the specified chaincode, which has been installed on the blockchain. After the simulation, the endorser cryptographically signs a message called `endorsement`, which contains `readset` and `writeset` (together with metadata such as transaction ID, endorser ID, and endorser signature) and sends it back to the client in a *"proposal response"*. The client collects endorsements until they satisfy the endorsement policy of the chaincode, which the transaction invokes. In particular, this requires all endorsers as determined by the policy to produce the same execution result (i.e., identical readset and writeset). Then, the client proceeds to create the transaction and passes it to the ordering service.

Ordering Phase

The ordering phase establishes a total order on all submitted transactions per channel. In other words, ordering atomically broadcasts endorsements and thereby establishes consensus on transactions, despite faulty orderers. Moreover, the ordering service batches multiple transactions into blocks and outputs a hash-chained sequence of blocks containing transactions. This phase is executed after a client has collected

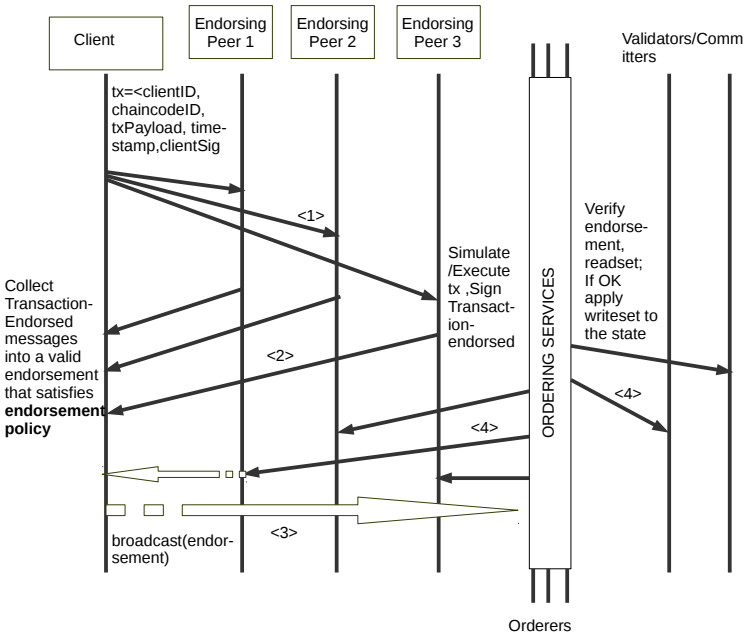


Figure 2.1: Transaction flow in Hyperledger Fabric v1.1

enough endorsements on a proposal. It assembles a transaction and submits this to the ordering service.

Validation Phase

Blocks are delivered to peers either directly by the ordering service or through gossip. A new block then enters the validation phase which consists of three sequential steps:

- **Endorsement policy evaluation** - It occurs in parallel for all transactions within the block and is responsible for validating the endorsement with respect to the endorsement policy configured for the chaincode.
- **Read-write conflict check** - This is done for all transactions in the block sequentially. For each transaction it compares the versions of the keys in the readset field to those in the current state of the ledger, as stored locally by the peer, and ensures they are still the same. In case of mismatch, transaction is marked invalid.
- **The ledger update phase** - Block is appended to the locally stored ledger and the blockchain state is updated. A bit mask is maintained denoting the transactions that are valid within the block. This facilitates the reconstruction of the state at a later time. Furthermore, all state updates are applied by writing all key-value pairs in writeset to the local state.

2.2 Related Work

2.2.1 Ring Signature

When a user with secret key $\mathcal{SK}$ wishes to generate an anonymous signature on a message $\mathcal{M}$, he chooses a ring $\mathcal{R}$ of public keys which includes his own, computes $\sigma \leftarrow Sign_{\mathcal{SK}}(\mathcal{M}, \mathcal{R})$ and outputs $(\sigma, \mathcal{R})$. Anyone can now verify that this signature was generated by someone holding a key in $\mathcal{R}$. A comparative study of various ring signature schemes, including our proposed scheme, has been made in Table 2.1.

Even though hiding identity using ring signature is a quite well studied area [8],[9],[33],[12], it's applicability in cryptocurrencies and blockchain is being recently explored. Monero, an anonymous cryptocurrency, improves on its existing *Cryptonote* [38] protocol by using a new efficient Ring Confidential Transactions protocol - `RingCT 2.0` [35]. It is based on the well-known Pedersen commitment, accumulator with one-way domain and signature of knowledge related to the accumulator. Over here, the size of signature is independent to the number of groups of input accounts in a transaction.

2.2.2 Threshold Signature

It is definitely better to distribute the storage of the cryptographic key for authorizing transactions among various parties instead of just trusting a single entity. Thus threshold signature scheme is preferred in blockchain systems. In a threshold signature scheme, the ability to construct a signature is distributed among n participants, each of whom receives a secret share of the private signing key. The participation of t or more of them is required to sign (for some fixed $t \leq n$). Various threshold signature scheme, as seen for Bresson et al. [6] in Table 2.1, involves a third party which is responsible for signature generation. Some are quiet complex to implement practically with signature size being dependent on the ring size(as in Yuen et al. [40]).

Several threshold signature schemes for enhancing Bitcoin security has been proposed. Goldfeder et al. [18] presented the first threshold signature scheme compatible with Bitcoin's `ECDSA` signatures and show how distributed Bitcoin wallets can be built using this primitive. Gennaro et al. [17] presented a threshold DSA algorithm with a similar motive of of securing Bitcoin wallets. Another scheme build on distributed Schnorr signatures to enhance the security and performance of Bitcoin by Kogias et al. [27] has been studied. They introduced `ByzCoin`, a cryptocurrency that replaces the proof-of-work used to reach a consensus in Bitcoin with a dynamic version of the `PBFT` protocol to achieve strong consistency.

*Exponentiation operation is of the form g^a for base g, multibase exponentiation operation is of the form $g^a.h^b$ for base g, h.

Table 2.1: Comparison of Ring Signature Schemes

Scheme	Signature Size	Security Notions	Linking Complexity	Signing * Complexity	Verify * Complexity	Problem Encountered
1-out-of-n Ho. Au *et al.* [4]	$\mathcal{O}(1)$	Unforgeability, Linkable Anonymity, Linkability, Non slanderability - all wrt adversarially chosen keys	$\mathcal{O}(1)$: check linkability tag group oriented linkability	Uses Signature based on Proof of Knowledge, (n+2) exponentiation and 7 multibase exponentiation	7 multibase exponentiation	Adversary can corrupt a member the signer $\mathcal{S}$ overhead of certificate check, need event oriented linkability since ring members are fixed
t-out-of-n[6] Bresson *et al.*	$\mathcal{O}(l.2^t n \log n)$	Unforgeability, *t-CMA* secure Anonymity	unlinkable	$t.2^t \log n + t$ symmetric cipher operation, $n.2^t \log n + n$ exponentiation	$t.2^t \log n$ symmetric cipher op, $t.2^t \log n$ exponentiation	Prover may be malicious, t signers need to share their need secret keys, computationally expensive, double signing can't be prevented
t-out-of-n Tsang *et al.*[37]	$\mathcal{O}$(n)	Unforgeability, Linkable Anonymity, Event-oriented Linkability, Non slanderability.	$\mathcal{O}(n^2)$ Event-oriented Linkability	2(n+d) exponentiations and 2(n-d) multibase exponentiation	$\mathcal{O}$(n) multibase exponentiations	Prover/Signer may create 2 different event-id(double signing possible), Problem of CDS [12] scheme exist, sharing of secret key with prover $\mathcal{P}$,when $\mathcal{P}$ gets compromised, is not desired
1-out-of-n[40] Yuen *et al.*	$\mathcal{O}(\sqrt{n})$	Unforgeability, linkable anonymity event-oriented linkability, non-	$\mathcal{O}(1)$, event-oriented linkability	(8+4$\sqrt{n}$) exponentiation, (4+2$\sqrt{n}$) multibase exponentiation One-time signature	(8+8$\sqrt{n}$) pairing, 2 exponentiation, one-time verification	Dependency on event-id, ,complex verification mechanism, signature complexity is high
t-out-of-n[40] Yuen *et al.*	$\mathcal{O}(t.\sqrt{n})$	Unforgeability, linkable anonymity event-oriented linkability, non-slanderability	$\mathcal{O}(t \log t)$,event-oriented linkability	(8t+4t$\sqrt{n}$) exponentiation, (4t+2t$\sqrt{n}$) multibase exponentiation	(8t+8t$\sqrt{n}$) pairing, 2t exponentiation, t one-time verification	More complex than *1-out-of-n* signature scheme
URS Franklin,Zhang[16], [29]	$\mathcal{O}$(n)	Unforgeability, secure linkability, and restricted anonymity	$\mathcal{O}(1)$ - tag is hash of message, ring members and private key of signer	2n-1 multibase exponentiation and 1 exponentiation	2n multibase exponentiation	Computationally expensive scheme, not yet extended to *t-out-of-n* scheme.
Our proposed signature scheme *FCsLRS*	$\mathcal{O}(1)$, constant size	Unforgeability Linkability, Linkable Anonymity, Non-slanderability, -all wrt adversarially chosen keys	$\mathcal{O}(1)$, transaction oriented linkability	11 exponentiations and 5 multibase exponentiations	10 multibase exponentiations and 6 exponentiation	Extension to *t-out-of-n* signature scheme not efficient.

2.2.3 Zero Knowledge Proofs

Zero-knowledge proofs allow one party (prover) to prove to another (verifier) that a statement is true, without revealing any information beyond the validity of the statement itself. Zero-knowledge proofs which are short (or "succinct" with a proof length of only a few hundred bytes even for statements about programs that are very large) can be verified within a few milliseconds. In interactive zero-knowledge protocols, the prover and verifier had to communicate back and forth for multiple rounds, but in "non-interactive" constructions, the proof consists of a single message sent from prover to verifier. This is achieved by generation of a common reference string shared between prover and verifier. Such proof constructions are referred to as *Zero-Knowledge Succinct Non-Interactive Argument of Knowledge* or zk-SNARK [1].

ZCash, another popular cryptocurrency secured by zero knowledge succinct non-interactive arguments of knowledge (zk-SNARKs), requires a trusted set up stage, but after that the system is entirely anonymous [21]. Hardjono, et al. [20] presented a new architecture called *ChainAnchor*, address the issue of retaining user anonymity, bringing in the concept of *semi-permissioned* blockchains. ChainAnchor builds upon and makes use of the zero knowledge proof mechanisms of the `EPID` scheme, which has the advantage of an optional cryptographic binding to a `TPM` tamper-resistant hardware. When a user seeks to prove that he or she is a group member to a verifier, it uses the *Camenisch-Lysyanskaya (CL)* signature which is based on "signature of knowledge" (as defined in 7).

2.2.4 Threshold Cryptography in Permissioned Blockchain

Till date, the only work which has focused on threshold signatures in permissioned blockchain - Hyperledger Fabric, is [34]. They have identified numerous potential application of threshold signature which can be used for group of Certificate Authorities, Byzantine Consensus protocols, chaincode applications and transaction validation. For this they have compared the performance of threshold signature schemes - threshold RSA signtaure/threshold BLS signature (ensures short signature size), against multisignature. However they have not focused on the problem of ensuring anonymity of endorsers. The main motive behind use of threshold signatures for transaction endorsement was to reduce the task of validator where verification of one signature instead of verifying each signature submitted by the pre-defined endorsers will serve the purpose.

Since our problem statement demands perfect anonymity on the identity of endorsers and not efficient transaction validation, we consider use of linkable ring signature for our anonymous endorsement system. It avoids all the complexities associated with the implementation of threshold cryptosystem. In the next chapter, we provide the detailed construction of the proposed signature scheme and extended it to implement threshold endorsement policy.

2.3 Preliminaries

2.3.1 Mathematical Notations

We first give some notations to be used in the rest of the thesis. N is an *RSA integer* if $N = pq$ for distinct primes p and q such that $|p| = |q|$. Let $\lambda, l, \mu \in \mathbb{N} : \lambda > l - 2, l/2 > \mu + 1$ be the security parameters and $\texttt{RSA}_\lambda$ be the set of RSA integers of size λ. A number p is a *safe prime* if $p = 2p' + 1$ and both p and p' are odd primes. A number N is a rigid integer if $N = pq$ for distinct safe primes p and q such that $|p| = |q|$. For $\lambda \in \mathbb{N}$, let $\texttt{Rig}_\lambda$ be the set of λ-bit rigid integers. $QR(N)$ denotes the group of quadratic residues modulo N.

2.3.2 Hardness Assumptions

- **Decisional Diffie-Hellman (DDH) Assumption.** [4] Let G be a group where $|G| = q$ and $g \in G$ such that $\langle g \rangle = G$. No PPT algorithm can, on input two distributions $\langle g, g^a, g^b, g^{ab} \rangle$ and $\langle g, g^a, g^b, g^c \rangle$ where $a, b, c \in_R \mathbb{Z}_q$, distinguish them with non-negligible probability over $1/2$ in time polynomial in q.

- **Strong RSA (SRSA) Assumption.** [4]There exists no probabilistic polynomial time (PPT) algorithm which, on input a random λ-bit safe prime product N and a random $z \in QR(N)$, returns $u \in \mathbb{Z}_N^*$ and $e \in \mathbb{N}$ such that $e > 1$ and $u^e = z (mod\ N)$, with non-negligible probability and in time polynomial in λ.

- **Link Decisional RSA (LD-RSA) Assumption.**[4] There exists no PPT algorithm which, on input a λ-bit safe prime product $N = p.q = (2p' + 1).(2q' + 1)$, $\hat{g} \in QR(N)$ with order $p'q', n_0 = p_0 q_0$ and $n_1 = p_1 q_1$ (where p_0, q_0, p_1, q_1 are sufficiently large random primes of size polynomial in λ), and $\hat{g}^{p_b + q_b}$ where $b \in_R \{0, 1\}$, returns $b' = b$ with probability non-negligibly over $1/2$ and in time polynomial in λ.

2.3.3 Building Blocks

Definition 1 ***(NP-Relations).*** *An **NP**-relation R is a relation over bitstrings for which there is an efficient algorithm to decide whether $(x, y) \in R$ in time polynomial is the length of x. The **NP-language** L_R associated to R is defined as $L_R = \{x | (\exists y)[(x, y) \in R]\}$*

Definition 2 ***(Σ-Protocols.)*** *[13] A Σ-protocol (Fig. 2.2) for an **NP**-relation R is an efficient 3-round two-party protocol, such that for every input (x, y) to prover P and x to verifier V, the first round, initiated by P, yields a commitment message `COM`, subsequently V replies with a random challenge message `CH`. The last round by P concludes by sending response message `RES`. At the end of a run, V outputs a*

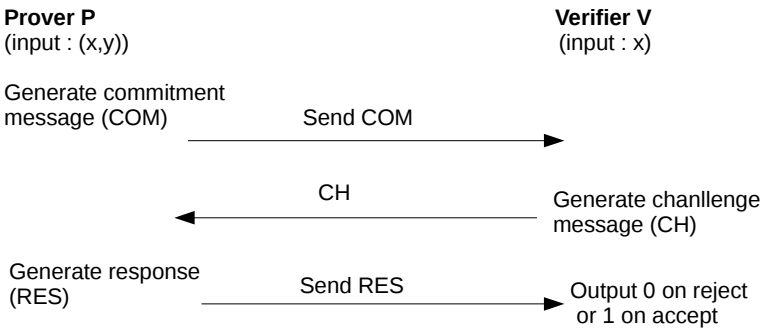


Figure 2.2: Execution of Σ-Protocol.

0/1 value, functionally dependent on x and the transcript π =(`COM,CH,RES`) only; a transcript is valid if the output of the honest verifier is 1.

A Σ-protocol satisfies :

- *Special Soundness*- there exists an efficient algorithm (called a *Knowledge Extractor*) that on input any $x \in L_R$ and any pair of valid transcripts with the same commitment message, (`COM, CH1, RES`$_1$) and (`COM, CH2, RES`$_2$) outputs y such that $(x, y) \in R$;
- *Special Honest-Verifier Zero-Knowledge* - there exists an efficient algorithm (called a *Simulator*) that on input $x \in L_R$ and any challenge message `CH`, outputs a pair of commitment/response messages `COM, RES` such that the transcript $\pi \doteq$ `(COM, CH, RES)` is valid, and it is distributed according to the probability distribution $(P(x, z) \leftrightarrow V(x))$, for any y such that $(x, y) \in R$.

Theorem 1 *A Σ-protocol for any **NP**-relation can be constructed if one-way functions exist.*

Definition 3 *(**Knowledge extractor**). Let V outputs 1, (on input x, auxiliary input $\tilde{z}$ and random input r : $x \in L_R$, and $\tilde{z}, r \in \{0,1\}^*$) after interacting with prover specified by $P_{x,\tilde{z},r}$ with probability $p(x, \tilde{z}, r)$ and $\kappa(.)$ be error, $\kappa : \mathbb{N} \to [0, 1]$.*

If $p(x, \tilde{z}, r) > \kappa(|x|)$, then a probabilistic oracle machine K, on input x (same as that given to V) and access to oracle $P_{x,\tilde{z},r}$, outputs a solution $y \in R(x)$ within an expected number of steps bounded by

$$\frac{q(|x|)}{p(x, \tilde{z}, r) - \kappa(|x|)}$$

*where q(.) is a positive polynomial. This oracle machine K is called a **(universal) knowledge extractor** ([19]).*

It can also be formulated alternatively as follows - A probabilistic oracle machine K is a knowledge extractor, if having access to oracle machine $P_{x,\tilde{z},r}$, runs in expected polynomial time and outputs a solution $s \in R(x)$ with probability atleast $\frac{p(x,\tilde{z},r)-\kappa(|x|)}{q(|x|)}$.

Definition 4 ***(PK-bijectivity).*** *Let $R \subseteq \mathcal{X} \times \mathcal{Y}$ be a one-way efficiently sampleable NP-relation. Let $\mathcal{X}_{\mathcal{R}} = \{x \in \mathcal{X} : \text{there exists } y \in \mathcal{Y}, (x,y) \in R\}$. A mapping $\theta : \mathcal{X}_R \to \mathcal{Z}$ is PK-bijective if the first two out of following three properties are satisfied. If all the three properties are satisfied, then it is special PK-bijective.*

1. *The mapping θ is one-way and bijective.*
2. *Let (x_0, y_0) and (x_1, y_1) be two random samples of R with $y_0 \neq y_1$. Let $b \in \{0,1\}$ be a fair coin flip and z=$\theta(x_b)$. Then there is no PPT algorithm which can, given z, distinguish between the two cases b=0 and b=1 with success probability non-negligibly over half.*
3. *Let (x_0, y_0) and (x_1, y_1) be two samples of R with $y_0 \neq y_1$. Let $b \in \{0,1\}$ be a fair coin flip and z=$\theta(x_b)$. Then there is no PPT algorithm which can, given z, distinguish between the two cases b=0 and b=1 with success probability non-negligibly over half.*

Definition 5 ***(Sphere Truncations of Quadratic Residues).*** *Given that N is a RSA Modulus integer where $N = pq$ and $p = 2p' + 1$ and $q = 2q' + 1$ with p, q, p', q' all are prime. Let $S(2^l, 2^\mu) = \{2^l - 2^\mu + 1, \ldots, 2^l + 2^\mu - 1\}$ be a sphere for two parameters $l, \mu \in N$ where $|S(2^l, 2^\mu)| = 2^{\mu+1} - 1$. Assuming factoring is hard and the fact the sphere $S(2^l, 2^\mu)$ is sufficiently large (but still not very large), then the random variable a^x with $x \in RS(2^l, 2^\mu)$ is indistinguishable from the uniform distribution over $QR(n)$. Intuitively, this means that a truncation of the QR(N) as defined by the sphere $S(2^l, 2^\mu)$ is indistinguishable to any probabilistic polynomial-time observer.*

Definition 6 ***(Discrete-log Relation Sets).*** *A discrete-log relation set R with z relations over r variables and m objects is a set of relations defined over the objects $A_1, \ldots, A_m \in G$, G is an unknown order group, and free variables $\alpha_1, \ldots, \alpha_r$ with the following specifications :*

1. *The i-th relation in the set R is specified by a tuple $\langle a_1^i, \ldots, a_m^i \rangle$ so that each a_j^i is selected to be one of the free variables $\{\alpha_1, \ldots, \alpha_r\}$ or an element of $\mathbb{Z}$. The relation is to be interpreted as $\Pi_{j=1}^m A_j^{a_j^i} = 1$.*
2. *Every free variable α_j is assumed to take value in a finite integer range $S(2_j^l, 2_j^\mu)$ where $l_j, \mu_j \geq 0$.*

Such sets are quite useful in planning complex proofs of knowledge for protocols operating over groups of unknown order in general like for group $QR(N)$.

A *discrete-log* relation set R is said to be *triangular*, if for each relation i involving the free variables $\alpha_1, \alpha_2, \ldots \alpha_k$, it holds that the free-variables $\alpha_1, \alpha_2, \ldots \alpha_k$ are contained in relations $1, \ldots, i-1$.

Definition 7 ***(Signature of Knowledge).*** *Every three-round Σ-protocols or Proof of Knowledge protocols (PoKs) that is Honest-Verifier-Zero-Knowledge(HVZK) can be transformed into a signature scheme by setting the challenge to the hash value of the commitment concatenated with the message to be signed [15]. Such signature schemes generated are provably secure [32] against existential forgery under adaptively chosen message attack in the random oracle model [5]. It is referred to as Signatures based on Proofs of Knowledge or SPK [9].*

As an example, consider $SoK\{(x) : y = g^x\}(\mathrm{m})$, where m is the message, the signature scheme derived from the zero-knowledge proof of the discrete logarithm of y using the above technique. Let R be a fixed NP-hard relation with the corresponding language $L = \{y : \exists\, \mathrm{x} \text{ such that} (x, y) \in R\}$. A relation is called hard if it is infeasible for any efficient algorithm, given some instance y, to compute a valid witness such that $(x, y) \in R$. It's security follows from the definition given in [11].

Definition 8 ***(Accumulators with One-Way Domain).*** *An accumulator family is a pair $(\{F_\lambda\}_{\lambda\in\mathbb{N}}, \{X_\lambda\}_{\lambda\in\mathbb{N}})$, where $(\{F_\lambda\}_{\lambda\in\mathbb{N}}$ is a sequence of families of functions such that each $f \in F_\lambda$ is defined as $f : U_f \times X_f^{ext} \leftarrow U_f$ for some $X_f^{ext} \subseteq X_\lambda$ and additionally the following properties are satisfied:*

- *(efficient generation) There exists an algorithm G that on input a security parameter 1^λ outputs a random element f of F_λ, possibly together with some auxiliary information a_f.*
- *(efficient evaluation) Any $f \in F_\lambda$ is computable in time polynomial in λ.*
- *(quasi-commutativity) For all $\lambda \in \mathbb{N}$, $f \in F_\lambda$, $u \in U_f, x_1, x_2 \in X_\lambda$,*

$$f(f(u, x_1), x_2) = f(f(u, x_2), x_1) \tag{2.1}$$

$\{X_\lambda\}_{\lambda\in\mathbb{N}}$ is referred to as the value domain of the accumulator. For any $\lambda \in \mathbb{N}, f \in F_\lambda$ and $X = \{x_1, \ldots, x_s\} \subset X_\lambda$, $f(\ldots f(u, x_1), \ldots, x_s)$ is the accumulated value of the set X over u: due to quasi-commutativity, such value is independent of the order of the $x_i's$ and will be denoted by $f(u, X)$.

Based on the Strong RSA assumption, an accumulator with one-way domain[4] is a quadruple $(\{F_\lambda\}_{\lambda\in\mathbb{N}}, \{X_\lambda\}_{\lambda\in\mathbb{N}}, \{Z_\lambda\}_{\lambda\in\mathbb{N}}, \{R_\lambda\}_{\lambda\in\mathbb{N}})$, such that the pair $(\{F_\lambda\}_{\lambda\in\mathbb{N}}, \{X_\lambda\}_{\lambda\in\mathbb{N}})$ is a collision-resistant accumulator, each R_λ is a relation over $X_\lambda \times Z_\lambda$ with the following properties: (efficient verification). There exists an efficient algorithm D that on input $(x, z) \in X_\lambda \times Z_\lambda$, returns 1 if and only if $(x, z) \in R_\lambda$. (efficient sampling). There exists a probabilistic algorithm W that on input 1^λ returns a pair $(x, z) \in X_\lambda \times Z_\lambda$

such that $(x, z) \in R_\lambda$, z *is the pre-image of* x. *(one-wayness). It is computationally hard to compute any pre-image* z' *of an* x *that was sampled with* W. *Formally, given a negligible value* $\nu(\lambda)$, *for any adversary* $\mathcal{A}$:

$$Pr[(x, z) \xleftarrow{R} W(1^\lambda); z' \xleftarrow{R} \mathcal{A}(1^\lambda, x) \mid (x, z') \in R_\lambda] = \nu(\lambda) \tag{2.2}$$

For $\lambda \in \mathbb{N}$, *the family* F_λ *consists of the exponentiation functions modulo* λ*-bit rigid integers :*

$$\begin{array}{c} f : QR(N) \times \mathbb{Z}_{N/4} \to QR(N) \\ f : (u, x) \to u^x \mod N \end{array} \tag{2.3}$$

where $N \in \mathtt{Rig}_\lambda$.

The accumulator domain $\{X_\lambda\}_{\lambda\in\mathbb{N}}$ *is defined by:*

$$X_\lambda = \{e \text{ prime} \mid (\frac{e-1}{2} \in \mathtt{RSA}_l) \wedge (e \in S(2^l, 2^\mu))\} \tag{2.4}$$

where $S(2^l, 2^\mu)$ *is the integer range* $(2^l - 2^\mu, 2^l + 2^\mu)$ *that is embedded within* $(0, 2^\lambda)$ *with* $\lambda - 2 > l$ *and* $l/2 > \mu + 1$. *The fact that the sphere* $S(2^l, 2^\mu))$ *is sufficiently large so that any random variable* $u^x \mod N$ *is indistinguishable from the uniform distribution over* $QR(N)$ *(from Definition 5. Sphere Truncation of Quadratic Residue). The pre-image domain* $\{Z_\lambda\}_{\lambda\in\mathbb{N}}$ *and the one-way relation* $\{R_\lambda\}_{\lambda\in\mathbb{N}}$ *are defined as follows:*

$$Z_\lambda = \left\{ \begin{array}{c} (e_1, e_2) \mid e_1, e_2 \text{ are distinct } l/2 - \text{bit} \\ \text{primes and } e_2 \in S(2^{\frac{l}{2}}, 2^\mu) \\ R_\lambda = \{(x, (e_1, e_2)) \in X_\lambda \times Z_\lambda \mid (x = 2e_1e_2 + 1)\} \end{array} \right\} \tag{2.5}$$

Chapter 3

Proposed Anonymous Endorsement System

To address the problem of biased endorsement policy as well as ensuring privacy of endorsers, we have designed an anonymous endorsement system. To ensure privacy of the system, we have proposed a new ring signature scheme which is discussed in Section 3.1.

In Hyperledger Fabric, membership service provider(MSP) identifies the parties, who are the members of a given organization in the blockchain network. The endorsement set for a particular chaincode is presumed to be predefined and remains fixed for a long time, unless any of the members get revoked. The right measure of "signature size" constructed for each transaction must not involve explicit description of the ring members(endorsers for this case). Thus for our proposed signature scheme, both the signer and the verifier need to perform a one-time computation of accumulation of public keys proportional to the size of the ring and get some constant-size information which allows them to generate or verify many subsequent signatures in constant time.

Entities present in the network

- *Fabric CA(Certification Authority) Server* * issuing enrolment certificates to all the peer nodes (endorser and validators). Setup mentioned in [22].

- *Client* : An entity lying outside the blockchain network, having a transaction request. A peer node, within the network, acts as a proxy for the client node.

- *Endorser set* $\mathcal{E}$: A pre-defined set to be specified before instantiation of chaincode. Members of this set, based on a given endorsement policy, decides on whether to endorse a transaction.

*It is a private root CA provider capable of managing digital identities of Fabric participants that have the form of X.509 certificates

- *Signer* S : A member of the endorsement set $\mathcal{E}$ which executes the ring signature algorithm on the transaction response packet for the endorsed transaction.
- *Verifier/Validator set* $\mathcal{V}$: Validator nodes verify whether the signature was generated by a valid member of the endorsement set.

Requirement of the signature scheme

- Signature and tag generated must be of short.
- The signature generation and verification must be computationally efficient.
- None of the entities must get access to any secret of the signer.
- Signatures generated according to specification are **accepted** during verification, with overwhelming probability.
- Two signatures signed according to specification are **linked** with overwhelming probability, if the two signatures are generated by the same signer on the same transaction for the same set of ring members.

3.1 Proposed construction of Fabric's Constant-Sized Linkable Ring Signature (FCsLRS)

In this section, we propose a new constant-sized *Linkable Ring Signature* scheme called as *Fabric's Constant-Sized Linkable Ring Signature*(FCsLRS) for a fixed set of ring members and discuss the construction details. Our construction is inspired by the signature scheme obtained by applying *Fiat-Shamir transformation* to the `Identification Protocol` suggested in Dodis et al.[13]. Previously, this identification protocol has been used as a short signature scheme by Tsang, et al.[36] and Ho Au, et al.[4] for e-Cash, e-voting and attestation. But none of them could have been used directly for our endorsement system.

Considering n to be the number of members in the endorsement set and t to be the threshold value. FCsLRS is represented as a tuple (`Init, KeyGen, AccumulatePubKey, GeneratePubKeyWitness, Sign, Verify, Link`) of seven polynomial time algorithms, which has been described below in details.

- `Init`. On input security parameter 1^λ, *Fabric CA (Certificate Authority)* prepares a collision-resistant accumulator with one-way domain, together with its description given above denoted by `desc`. A generator $u \in QR(N)$ ($|QR(N)| = p'q' = \phi(N)/4$, where $\phi(N) = (p-1)(q-1)$ is Euler's totient)is picked up uniformly at random, where $N \in \mathtt{Rig}_\lambda$ and outputs the system parameters `param` as $(1^\lambda, \mathtt{desc}, u)$. Public parameters $g, h, y, t, s, \zeta \in QR(N)$, is also generated. These parameters remain same across all the transactions.

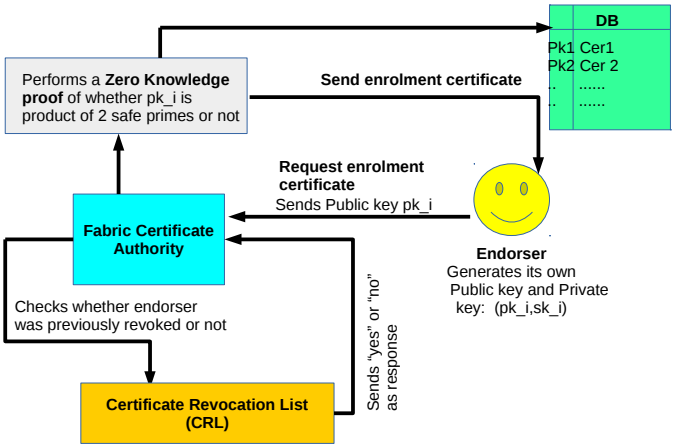


Figure 3.1: PKI

- `KeyGen`. On input the system's parameters `param`, the algorithm parses it into $(1^\lambda, \texttt{desc}, u)$. Key pair (sk_i, pk_i) is generated for each endorser $E_i \in \mathcal{E}, 1 \leq i \leq n$ by executing the probabilistic sampling algorithm W of their accumulator[†] to obtain $(pk_i, sk_i) = (y_i, (p_i, q_i))$ such that y_i is a prime number and and both p_i and q_i is of length (in binary) approximately $l/2$ bits. Also, $q_i \in S(2^{l/2}, 2^\mu)$. Upon obtaining the key pair, endorser E_i submits its public key y_i and *verifiable credentials* to Fabric CA. The CA first checks whether such credentials matches with any of those present in *Certificate Revocation List(CRL)*. If yes, then its *enrolment certificate* was previously revoked and hence cannot be added as a network entity. Else, E_i proves in zero-knowledge to CA that $(y_i - 1)/2$ is a product of two primes of the same size. An interactive two-party protocol called `Prove` [7], [8]. If endorser is able to prove, then CA issues an *enrolment certificate* to it. The identity of the endorser, public keys $y_j, 1 \leq j \leq n$ along with enrolment certificate gets added to the public database $\mathcal{DB}$ (any valid entity in the network has access to this database).

- `AccumulatePubKey`. Fabric CA executes this algorithm for combining all the public keys in public database $\mathcal{DB}$. The accumulated value v calculated by

[†]All endorsers run the sampling algorithm of the accumulator in parallel

using data from $\mathcal{DB}$, is :

$$\begin{aligned} v &= f(u, \{y_j | 1 \leq j \leq n\}) \\ &= f(f(\ldots f(u, y_1), y_2), y_3) \ldots, y_n) \\ &= f(f(\ldots f(u^{y_1} \mod N, y_2), y_3), \ldots y_n) \\ &= f(f(\ldots f((u^{y_1} \mod N)^{y_2} \mod N, y_3) \ldots), y_n) \\ &\quad \ldots\ldots \\ &\quad \ldots\ldots \\ &= (\ldots(((u^{y_1} \mod N)^{y_2} \mod N)^{y_3} \mod N) \ldots)^{y_n} \mod N) \end{aligned} \tag{3.1}$$

This value is generated and used for long time unless the endorsement set $\mathcal{E}$ changes. Hence the computation can be said to be performed one time before instantiation of chaincode in all the peer nodes of the network.

- `GeneratePubKeyWitness`. Each member e of set $\mathcal{E}$ computes witness $w_e \leftarrow f(u, \{y_e | 1 \leq i \leq n, i \neq e\}), \langle u \rangle = QR(N)$ for public key y_e , where accumulated value v can be generated by computing $v \leftarrow f(w_e, y_e)$. When the endorser is willing to endorse or sign a transaction, it uses this value w_e for construction of Signature based on Proof of Knowledge. As we have considered endorsement set to be fixed, even this value can be pre-computed.

- `Sign`. Endorser $E_\pi \in \mathcal{E}$ who wants to endorse a transaction is the Signer $\mathcal{S}$. It obtains the public key set $\mathcal{DB} = \{y_1, y_2, \ldots, y_n\}$, posessing a valid *enrolment certificate* and has not been revoked (CA performs the check and informs if any endorser has been put in *CRL*).

  A new linking criterion called *Transaction-Oriented linkability* has been used in which one can tell if two signatures are linked if and only if they are signed by a common signer for a given transaction (similar to the concept of *Event-oriented* linkability in [37]). For this purpose we use a public parameter g_{tid} instead of simply using $g \in QR(N)$. To construct g_{tid}, we consider $g \in QR(N)$ and a function $\tilde{H} : \mathbb{N} \rightarrow \mathbb{G}$, $\mathbb{G} \subset \mathbb{Z}_{N/4}$ which generates $\tilde{tx} = \tilde{H}$(transaction-id), where *transaction-id* is unique for each transaction, which is again the hash of the transaction payload `txPayload`. Thus, $g_{tid} = f(g, \tilde{tx}) = g^{\tilde{tx}} \mod N$, where f is the function defined as in Eq. 2.3.

  For a given message $m \in \mathcal{M}$ (which is the *transaction-response*) which has a transaction id *transaction-id*, a private key $sk_\pi = (p_\pi, q_\pi)$ that corresponds to original public key, y_π, accumulated value v and secret value w_π, signer $\mathcal{S}$ does the following :

  – $\mathcal{S}$ computes a signature for (notations used as per [4])

$$SPK\left\{\begin{pmatrix} w_\pi, y_\pi \ , \\ p_\pi, q_\pi \end{pmatrix} : \begin{array}{c} w_\pi^{y_\pi} = v \mod N \wedge y_\pi = 2p_\pi q_\pi + 1 \wedge \\ y_\pi \in S(2^l, 2^\mu) \wedge q_\pi \in S(2^{\frac{l}{2}}, 2^\mu) \wedge \\ \tilde{y} = \theta_d(p_\pi, q_\pi) : \theta_d(p_\pi, q_\pi) = g_{tid}^{p_\pi + q_\pi} \mod N, \end{array}\right\}(m) \tag{3.2}$$

where θ_d is a one-way bijective mapping and $\tilde{y}$ is the tag generated corresponding to the signature.

Signature based on proof of knowledge is basically a signature scheme in which a signer can speak on behalf of any **NP** statement (as stated in 3.2) to which he knows a witness/es without revealing all the irrelevant information [11]. Here the witness values are w_π, y_π, p_π and q_π. Any person who knows a satisfying assignment (that means posses the knowledge of witness) to the statements (in 3.2) has signed the message.

A practical Σ-protocol for relation stated in Eq. 3.2 is constructed using the framework of discrete logarithm sets, which allows to construct complex proofs of knowledge over groups of unknown order ($\phi(n)$ cannot be computed by any member of endorsement set $\mathcal{E}$). The public parameters $g_{tid}, h, y, t, s, \zeta \in QR(N)$ with unknown relative discrete logarithms alongwith the sequence of public values T_1, T_2, T_3, T_4, T_5 such that

$$T_1 = g_{tid}^r \ , T_2 = h^r \zeta^{x+r} \ , T_3 = s^r g_{tid}^{e_2} \ , T_4 = wy^r \ , T_5 = t^r g_{tid}^{2e_1}$$

where $r \xleftarrow{R} [0, \lfloor N/4 \rfloor - 1]$ is used for the construction of proof.

The public values T_1 is for the free variable r, T_2 is for the free variable x, T_3 is for the free variable e_2, T_4 is for the free variable w and T_5 is for the free variable e_1. Note that all the construction from T_2 to T_5 satisfy the property of triangularity with respect to first relation T_1. The construction of T_2 cannot be chosen of the form $h^r g_{tid}^x \mod N$ since x belongs to the set $\mathcal{DB}$ whose size is negligible compared to the size $S(2^{\frac{l}{2}}, 2^\mu)$, exponential order of the security parameter l. If prover P sends this value of T_2 to verifier $\mathcal{V}$, it can figure out, in polynomial time, the public key of the endorser/signer during verification phase. The **NP** statements used for generating Signature based on Proof of Knowledge is given below :

$$\begin{gathered}
T_1 = g_{tid}^r, \text{ (witness of } r) \\
T_2 = h^r.\zeta^r.\zeta^x = h^r.\zeta^{r+x}, \text{ (witness of } x \in S(2^l, 2^\mu)) \\
(T_1)^x = g_{tid}^{a_1}, \text{ (witness of } a_1) \\
(T1)^{e_2} = g_{tid}^{a_2}, \text{ (witness of } a_2) \\
T_3 = s^r g_{tid}^{e_2}, \text{ (witness of } e_2 \in S(2^{l/2}, 2^\mu)) \\
(T_4)^x = vy^{a_1}, \text{ (witness of } x) \\
(T_5)^{e_2} g_{tid} = t^{a_2}.g_{tid}^x \text{ (witness of } e_2 \text{ being non-trivial factor of } x) \\
(T_3)^2 T_5 = s^{2r}.t^r.\tilde{y}^2 \text{ (correctness of } \tilde{y})
\end{gathered} \tag{3.3}$$

for the free variables r, x, e_2, a_1, a_2 such that $x \in \mathcal{S}(2^l, 2^\mu), e_2 \in S(2^l, 2^\mu), a_1 = rx$ and $a_2 = re_2$. The signer $\mathcal{S}$ gives a proof of knowledge for witness w, y_π, p_π and q_π by satisfying the above equations. The variables x, e_1 and e_2 is assigned

value y_π (public key of $\mathcal{S}$), p_π and q_π respectively. A proof of the validity of the above 8 statements ensures that the prover knows a witness w for some value x in the accumulated value v and for the same x, the value $x-1$ can be split by the prover into two integers one of which belongs to $S(2^{l/2}, 2^\mu)$. This latter range-property guarantees the non-triviality of the splitting i.e., that the prover knows a non-trivial factor of $x-1$ (i.e., different than -1, 1, 2). The security parameters l, μ, ϵ, k must selected so that $l/2 > \epsilon(\mu + k) + 2$.

Public Parameters : $g_{tid}, h, t, s, y, \zeta, v \in QR(N), T_1, T_2, T_3, T_4, T_5, \mathbb{Z}_{N/4} \subset S(2^l, 2^\mu)$.

1. `Signer` $\mathcal{S}$ computes

$$\begin{aligned} \alpha_1 &\xleftarrow{R} \mathbb{Z}_{N/4}, \\ \alpha_2 &\xleftarrow{R} \mathbb{Z}_{N/4}, \\ \alpha_3 &\xleftarrow{R} \mathbb{Z}_{N/4}, \\ u_1 &\leftarrow g_{tid}^{\alpha_1} \ mod\ N, \\ u_2 &\leftarrow \zeta^{\alpha_1+\alpha_2} \ mod\ N, \\ u_3 &\leftarrow h^{\alpha_1} \ mod\ N, \\ u_4 &\leftarrow g_{tid}^{\alpha_1} \ mod\ N, \\ u_5 &\leftarrow g_{tid}^{\alpha_3} \ mod\ N, \\ u_6 &\leftarrow w^{\alpha_2} \ mod\ N, \\ u_7 &\leftarrow g_{tid}^{2e_1.\alpha_3} \ mod\ N, \\ u_8 &\leftarrow t^{\alpha_1} \ mod\ N \\ u_9 &\leftarrow g_{tid}^{\alpha_2} \ mod\ N \end{aligned} \tag{3.4}$$

2. $\mathcal{S}$ computes $c = H_1(m||u_1||u_2||u_3||u_4||u_5||u_6||u_7||u_8||u_9), H_1 : \mathcal{M} \times QR(N)^8 \rightarrow \mathcal{C}, \mathcal{C} \subseteq QR(N)$ and uses it to compute

$$\begin{aligned} \tilde{\alpha_1} &\leftarrow \alpha_1 + c.r, \\ \tilde{\alpha_2} &\leftarrow \alpha_2 + c.x, \\ \tilde{\alpha_3} &\leftarrow r.\alpha_2 + r.c.x, \\ \tilde{\alpha_4} &\leftarrow \alpha_3 + c.e_2, \\ \tilde{\alpha_5} &\leftarrow r.\alpha_3 + r.c.e_2 \end{aligned} \tag{3.5}$$

   $\mathcal{S}$ sends the signature $\sigma' = (u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, \tilde{\alpha_1}, \tilde{\alpha_2}, \tilde{\alpha_3}, \tilde{\alpha_4}, \tilde{\alpha_5}, \tilde{y})$ where $\tilde{y} = g_{tid}^{p_\pi + q_\pi}$ to all the validators in set $\mathcal{V}$.

`Signature size` : Values to communicated to the *Verifier* as Signature based on Proof of Knowledge are $\sigma = (u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, \tilde{\alpha_1}, \tilde{\alpha_2}, \tilde{\alpha_3}, \tilde{\alpha_4}, \tilde{\alpha_5}, \tilde{y})$, $u_1, u_2, \ldots, u_9, \tilde{\alpha_1}, \tilde{\alpha_2}$ and $\tilde{\alpha_5}$, each approximately being λ bits in size and $\tilde{\alpha_3}, \tilde{\alpha_5}$ each approximately being 2λ bits in size. Hence the signature generated is of constant size, being $\mathcal{O}(\lambda)$ where λ is the security parameter.

- `Verify`. To verify the signature $\sigma' = (u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, \tilde{\alpha_1}, \tilde{\alpha_2}, \tilde{\alpha_3}, \tilde{\alpha_4}, \tilde{\alpha_5}, \tilde{y})$ on message $m \in \mathcal{M}$, all the validator nodes in $\mathcal{V}$

computes $c = H_1(m||u_1||u_2||u_3||u_4||u_5||u_6||u_7||u_8||u_9)$,
$H_1 : \mathcal{M} \times QR(N)^8 \rightarrow \mathcal{C}, \mathcal{C} \subseteq QR(N)$ and checks if all the statements in Eq. 3.6 is valid or not. For *1-out-of-n* endorsement policy, if all check passes, then the verifier outputs `accept`; otherwise it outputs `reject` and aborts. For *t-out-of-n*, $t > 1$, we need to perform the test for signature linkability (`Link`) as well for final acceptance.

- `Link`. In [4], the tag generated is $\theta_d = ((e1, e2)) = g^{e_1+e_2}$. θ_d being PK-bijective, it prevented double signing on the same message. However as the endorsement set remains fixed, tag constructed must be function of the secret key as well as the transaction payload. For (*Transaction-Oriented linkability*), tag construction is modified by introducing transaction-id, a unique value associated with each transaction payload. Since $g_{tid} \in QR(N)$, modified $\theta_d = g_{tid}^{e_1+e_2}$ remains PK-bijective. Given two valid signatures σ'_1 and σ'_2 for a given transaction, validator node checks checks if $\tilde{y_1} = \tilde{y_2}$. If yes, output `linked`. Otherwise, output `unlinked`.

3.2 Extending to threshold endorsement policy

Given a threshold value $t, 1 \leq t \leq n/2$, if a validator node receives at least t out of n *transaction-response* with a valid, pairwise unlinked signatures (after $\binom{t}{2}$ tests of linkability) whose responses (read set and write set) are the same, then the endorsement policy is said to be satisfied. If each of at least $\frac{|\mathcal{V}|}{2}$ validator nodes in $\mathcal{V}$ reach a consensus on receipt of at least t signatures for the given transaction, then one of the honest validator node "broadcast" the *transaction-response* within a *transaction message* to the *Ordering Service* so that the transactions can be ordered chronologically by the channel.

$$g_{tid}^{\tilde{\alpha_1}} \stackrel{?}{=} u_1.T_1^c,$$
$$g_{tid}^{\tilde{\alpha_1}} \stackrel{?}{=} g_{tid}^{\alpha_1}.{g_{tid}^r}^c \mod N \ (\because Eq.\ 3.3, 3.4, 3.5),$$
$$g_{tid}^{\tilde{\alpha_1}} \stackrel{?}{=} g_{tid}^{\alpha_1+r.c} \mod N$$

$$\zeta^{\tilde{\alpha_2}+\tilde{\alpha_1}} h^{\tilde{\alpha_1}} \stackrel{?}{=} u_2.u_3.T_2^c,$$
$$\zeta^{\tilde{\alpha_1}+\tilde{\alpha_2}} h^{\tilde{\alpha_1}} \stackrel{?}{=} \zeta^{\alpha_1+\alpha_2}.h^{\alpha_1}.(h^r.\zeta^{x+r})^c \mod N,\ (\because Eq.\ 3.3, 3.4, 3.5),$$
$$\zeta^{\tilde{\alpha_1}+\tilde{\alpha_2}} h^{\tilde{\alpha_1}} \stackrel{?}{=} \zeta^{\alpha_2+\alpha_1+c.(x+r)}.h^{\alpha_1+r.c} \mod N$$

$$g_{tid}^{\tilde{\alpha_3}} \stackrel{?}{=} T_1^{\tilde{\alpha_2}},$$
$$g_{tid}^{\tilde{\alpha_3}} \stackrel{?}{=} (g_{tid}^r)^{\alpha_2+c.x} \mod N,\ (\because Eq.\ 3.3, 3.5),$$
$$g_{tid}^{\tilde{\alpha_3}} \stackrel{?}{=} g_{tid}^{r.\alpha_2+r.c.x} \mod N.$$

$$g_{tid}^{\tilde{\alpha_5}} \stackrel{?}{=} T_1^{\tilde{\alpha_4}},$$
$$g_{tid}^{\tilde{\alpha_5}} \stackrel{?}{=} (g_{tid}^r)^{\alpha_3+c.e_2} \mod N \ (\because of Eq. 3.3, 3.5),$$
$$g_{tid}^{\tilde{\alpha_5}} \stackrel{?}{=} g_{tid}^{r.\alpha_3+r.c.e_2} \mod N.$$

$$g_{tid}^{\tilde{\alpha_4}}.s^{\tilde{\alpha_1}} \stackrel{?}{=} T_3^c.u_4.u_5,$$
$$g_{tid}^{\tilde{\alpha_4}}.s^{\tilde{\alpha_1}} \stackrel{?}{=} (s^r g_{tid}^{e_2})^c.s^{\alpha_1}.g_{tid}^{\alpha_3} \mod N \ (\because Eq.\ 3.3, 3.4, 3.5), \tag{3.6}$$
$$g_{tid}^{\tilde{\alpha_4}}.s^{\tilde{\alpha_1}} \stackrel{?}{=} g_{tid}^{\alpha_3+c.e_2}.s^{\alpha_1+c.r} \mod N$$

$$u_6.v^c.y^{\tilde{\alpha_3}} \stackrel{?}{=} T_4^{\tilde{\alpha_2}},$$
$$w^{\alpha_2}.(w^x)^c.y^{\tilde{\alpha_3}} \stackrel{?}{=} (w.y^r)^{\tilde{\alpha_2}} \mod N \ (\because Eq.\ 3.3, 3.4, 3.5),$$
$$w^{\alpha_2+c.x}.y^{\tilde{\alpha_3}} \stackrel{?}{=} w^{\tilde{\alpha_2}}.y^{r.\tilde{\alpha_2}} \mod N$$

$$t^{\tilde{\alpha_5}}.g_{tid}^{\tilde{\alpha_2}}.u_7 \stackrel{?}{=} T_5^{\tilde{\alpha_4}}.u_9.g_{tid}^c$$
$$t^{\tilde{\alpha_5}}.g_{tid}^{\tilde{\alpha_2}}.g_{tid}^{2.e_1.\alpha_3} \stackrel{?}{=} (t^r.g_{tid}^{2.e_1})^{\tilde{\alpha_4}}.g_{tid}^{\alpha_2}.g_{tid}^c \mod N \ (\because Eq.\ 3.3, 3.4, 3.5),$$
$$t^{\tilde{\alpha_5}}.g_{tid}^{\tilde{\alpha_2}+2.e_1.\alpha_3} \stackrel{?}{=} t^{r.\tilde{\alpha_4}} g_{tid}^{2.e_1.\tilde{\alpha_4}+\alpha_2+c} \mod N$$
$$t^{\tilde{\alpha_5}}.g_{tid}^{\tilde{\alpha_2}+2.e_1.\alpha_3} \stackrel{?}{=} t^{r.\tilde{\alpha_4}} g_{tid}^{2.e_1.(\alpha_3+c.e_2)+\alpha_2+c} \mod N$$
$$t^{\tilde{\alpha_5}}.g_{tid}^{\tilde{\alpha_2}+2.e_1.\alpha_3} \stackrel{?}{=} t^{r.\tilde{\alpha_4}} g_{tid}^{2.e_1.\alpha_3+2.c.e_1.e_2+\alpha_2+c} \mod N$$
$$t^{\tilde{\alpha_5}}.g_{tid}^{\tilde{\alpha_2}+2.e_1.\alpha_3} \stackrel{?}{=} t^{r.\tilde{\alpha_4}} g_{tid}^{2.e_1.\alpha_3+c.(x-1)+\alpha_2+c} \mod N$$

$$\tilde{y}^{2c}.s^{2\tilde{\alpha_1}}.t^{\tilde{\alpha_1}} \stackrel{?}{=} (T_3^2.T_5)^c.u_4^2.u_8$$
$$\tilde{y}^{2c}.s^{2\tilde{\alpha_1}}.t^{\tilde{\alpha_1}} \stackrel{?}{=} ((s^r.g_{tid}^{e_2})^2.t^r.g_{tid}^{2.e_1})^c.(s^{\alpha_1})^2.t^{\alpha_1} \mod N (\because Eq.\ 3.3, 3.4, 3.5),$$
$$\tilde{y}^{2c}.s^{2\tilde{\alpha_1}}.t^{\tilde{\alpha_1}} \stackrel{?}{=} s^{2.r.c}.g_{tid}^{2.c.e_2}.t^{r.c}.g_{tid}^{2.e_1.c}.s^{2.\alpha_1}.t^{\alpha_1} \mod N$$
$$\tilde{y}^{2c}.s^{2\tilde{\alpha_1}}.t^{\tilde{\alpha_1}} \stackrel{?}{=} g_{tid}^{2.c.e_1+2.c.e_2}.s^{2.r.c+2.\alpha_1}.t^{r.c+\alpha_1} \mod N$$

Chapter 4

Security Model

4.1 Assumptions made

Some assumptions made regarding the entities in Hyperledger Fabric are :

- Fabric CA is honest.
- Endorsement set is fixed.
- All the peer nodes have their local copy of database consistent with the world state.
- Signature generation algorithm follows a *Random Oracle Model*.
- At least half of the verifiers and at least $n/2 + 1$ endorsers is honest.

The security model defined here is similar to the one defined in [36],[4].

4.2 Syntax

A *Linkable Ring Signature* scheme is a tuple (`Init, KeyGen, AccumulatePubKey, GeneratePubKeyWitness, Sign, Verify, Link`) of seven polynomial time algorithms. Instead of a single entity generating keys for all the participants in the permissioned blockchain, we define `KeyGen` as an algorithm executed by each individual user for the generation of the public and private key pair. Syntax is as follows :

- `param` $\leftarrow$ `Init`(1^λ), the poly-time *initialization* algorithm which, on input a security parameter $\lambda \in \mathbb{N}$, outputs the system parameters `param` containing, among other things, 1^λ. All other algorithms implicitly use λ as one of their inputs.

- $(sk_i, pk_i) \leftarrow$ `KeyGen()`, the PPT (*probabilistic polynomial time*) *key generation* algorithm which outputs a secret/public key pair (sk_i, pk_i). $\mathcal{SK}$ and $\mathcal{PK}$ denote the domains of possible secret keys and public keys respectively. All the generated $pk_i, 1 \leq i \leq n$ for n participants is made publicly available along with system parameter `param`.

- $(v) \leftarrow$ `AccumulatePubKey()`, the deterministic poly-time algorithm which, on input a set $\mathcal{Y}$ of n public keys in $\mathcal{PK}$, where $n \in \mathbb{N}$ is of size polynomial in λ, produces the value v.

- $(w) \leftarrow$ `GeneratePubKeyWitness()`, the deterministic poly-time algorithm which, on input a set $\mathcal{Y}' = \mathcal{Y} \setminus \{pk_e\}$ i.e. all public keys except that of the entity e who executes it ($n \in \mathbb{N}$ is of size polynomial in λ), produces the value w_e such that $f(w_e, pk_e) = w_e^{pk_e} = v$.

- For a *Signatures based on Proofs of Knowledge*, the Σ-protocol between signer and verifier for the **NP**-relation stated in Eq.3.2 has been converted into a signature scheme. It comprises the `(Sign,Verify)` algorithm pair, executed on the signer and verifier side respectively. Execution of this protocol is time independent from the number of public keys that gets aggregated in `AccumulatePubKey` or `GeneratePubKeyWitness`.

  - $\sigma \leftarrow$ `Sign`$(\mathcal{Y}, M, x)$, the PPT *signing* algorithm which, on input a set $\mathcal{Y}$ of n public keys in $\mathcal{PK}$, where $n \in \mathbb{N}$ is of size polynomial in λ, a message $M \in \{0,1\}^*$, and a private key $x \in \mathcal{SK}$ whose corresponding public key is contained in $\mathcal{Y}$, produces a signature σ. We denote by Σ the domain of possible signatures.
  - $1/0 \leftarrow$ `Verify`$(\mathcal{Y}, M, \sigma)$, the poly-time *verification* algorithm which, on input a set $\mathcal{Y}$ of n public keys in $\mathcal{PK}$, where $n \in \mathbb{N}$ is of size polynomial in λ, a message $M \in \{0,1\}^*$ and a signature $\sigma \in \Sigma$, returns 1 or 0 meaning `accept` or `reject` respectively. If the algorithm returns `accept`, the message-signature pair (M, σ) is said to be *valid*. The signature scheme must satisfy *Verification Correctness*, i.e. signatures signed by honest signer as per the specification must be accepted by an honest verifier with overwhelming probability.

- $1/0 \leftarrow$ `Link`(σ_0, σ_1), the poly-time linking algorithm which, on input two valid signatures, checks their corresponding tag and outputs 1 (if tags are same - signatures are linked) or 0 (if tags are different - unlinked signature) meaning `linked` or `unlinked` respectively. The signature scheme must satisfy *Linking Correctness*, i.e. any two signatures signed by a common honest signer on the same message are `linked` with overwhelming probability. On the other hand, any two signatures signed by two different honest signer must be `unlinked` with overwhelming probability.

4.3 Security Notions

Before defining the security notions, let us define the adversarial model and the possible attacks :

4.3.1 Adversarial Model

- Any corrupt endorser may launch insider attack (threat or use of influence) on the rest of the endorsers, acquiring their private keys
- Members (excluding the signer) belonging to the endorsement set may collude and reveal their secret keys on receipt of signature
- Validator may hold back the packets without verifying.
- Validator can act maliciously by randomly mark a transaction as valid/invalid without actually verifying.

Any adversary is assumed to have the following oracle access:

- $sk_i \leftarrow \mathcal{CO}(pk_i)$. The *Corruption Oracle,* on input a public key $pk_i \in \mathcal{Y}$ that is an output of `KeyGen()`, returns the corresponding secret key $sk_i \in \mathcal{X}$.
- $\mathcal{SO}(s, M, R)$. The *Signing Oracle,* on input a designated signer s, message M and subring R, where R is some subset of $\mathcal{Y}$ returns a valid signature σ which is computationally indistinguishable from one produced by `Sign(`$\mathcal{Y}, M, x$`)` using the real secret key x of signer s on message M.

Note, that if endorsement logic gets corrupted by adversary then it needs a mechanism of formal verification to check whether desired output is achieved or not. This is beyond our scope of work. Based on the last two points, we discus the correctness and soundness property of the Σ-protocol as well as security of signature scheme in the permissioned blockchain framework.

4.3.2 Correctness

For correctness, any execution of the Σ-protocol for the NP-relation given in Eq.3.2 will terminate with the verifier outputting 1, with overwhelming probability, if and only if a prover or an endorser possess the correct witness values $(y_\pi, w_\pi, p_\pi, q_\pi)$ for the corresponding accumulated public value v.

4.3.3 Soundness

The Honest-Verifier Zero-Knowledge property of the Σ-protocol for **NP**-relation Eq.3.2 guarantees that the transcript generated out of the interaction between signer and

verifier does not leak any information to the adversary $\mathcal{A}$ that has no knowledge of the secret. The soundness property is formalized in terms of the game played between Fabric CA and adversary $\mathcal{A}$, assuming all endorsers, participating in ring formation, are honest.

- Fabric CA runs the `Init` algorithm for security parameter λ and generates global parameters `param`. All endorsers executes `KeyGen` algorithm to generate the public key and private key pair and stored in public database $\mathcal{DB}$.
- $\mathcal{A}$ receives `param` from Fabric CA and gets the transcript of prior runs of the protocol between an honest signer and verifier. Given $\mathcal{A}$ has access to the corruption oracle $\mathcal{CO}$, it can query for the secret key of some but not all endorsers, who has put their public keys in database $\mathcal{DB}$.
- $\mathcal{A}$ now select a set of endorsers E' for which it has not queried their secret keys. It generates a value v' by accumulation of public keys of E'.
- $\mathcal{A}$ starts executing the Σ-protocol in the role of the signer and the probability of winning the game is negligible. Following the correctness property, an honest verifier with output 1(accept) with overwhelming probability if and only if the $\mathcal{A}$ can produce the correct secret value $(y_\pi, w_\pi, p_\pi, q_\pi)$ corresponding to accumulated value v'.

Note that if $\mathcal{A}$ is not given access to a correct tuple $(y_\pi, w_\pi, p_\pi, q_\pi)$ but still it wins the game, then it must have generated it by himself/herself. This contradicts the one-wayness of accumulator's domain.

4.3.4 Security Analysis of Signature Scheme

Since the architecture of *Fabric* is modular, the proposed signature scheme can be plugged-in as a feature. Given that Fabric is secure (Security model discussed in [23],[3]), we need to argue on the security of the proposed anonymous endorsement system based on the security of *FCsLRS* scheme.

Theorem 2 *If FCsLRS scheme is unforgeable, linkable anonymous, linkable and non-slandearable, then the scheme is secure and hence the proposed Anonymous Endorsement System also remains secure in the random oracle model.*

We have defined the security notions in details :

Unforgeability

The following construction of *constant-sized linkable ring signature* is unforgeable against "chosen subring" or "chosen public-key" attacks. The adversary is further allowed to corrupt endorsers and acquire their private keys i.e. it is unforgeable with respect to insider corruption.

Definition 9 ***(Unforgeability)****.[4] A linkable ring signature scheme is unforgeable if for any PPT adversary $\mathcal{A}$ and for any polynomial n(.), the probability that $\mathcal{A}$ succeeds in the following game is negligible:*

1. *(Initialization Phase.) Key pairs $(PK_i, SK_i)_{i=1}^{n(k)}$ are generated by executing KeyGen(1^k), and the set of public keys $S \doteq \{PK_i\}_{i=1}^{n(k)}$ is given to $\mathcal{A}$.*
2. *(Probing Phase.) $\mathcal{A}$ is given access to a signing oracle $\mathcal{SO}(.,.,.)$ where $\mathcal{SO}(s, M, R)$ outputs $Sign_{s,SK_s}(M, R)$ and it is required that $R \subseteq S$ and $PK_s \in R$. $\mathcal{A}$ is also given access to a corrupt oracle $\mathcal{CO}(.)$, where $\mathcal{CO}(i)$ outputs SK_i.*
3. *(Output Phase.) $\mathcal{A}$ outputs (M^*, σ^*, R^*), and succeeds if $\texttt{Verify}_{R^*}(M^*, \sigma^*) = 1, \mathcal{A}$ never queried $(., M^*, R^*)$ to its signing oracle, and $R^* \subseteq S \setminus C$, where C is the set of corrupted users.*

Linkable-Anonymity

A secure linkable ring signature cannot be anonymous against attribution attacks/key exposure attack if all the participants except signer gets corrupted and reveals their secret in order to frame the signer ([4]).

But if we consider *t-out-of-n* threshold endorsement policy , $1 \leq t \leq n/2$, where more than half of the endorsers in $\mathcal{E}$ is assumed to be honest, possibility of such attacks is negligible since atleast $n/2 + 1$ members in $\mathcal{E}$ will not reveal their secret keys .

Definition 10 ***(Linkable-anonymity)****.[4] A linkable ring signature is linkably anonymous if for any PPT adversary $\mathcal{A}$ and for any polynomial n(.), the probability that $\mathcal{A}$ succeeds in the following game is negligibly close to 1/2:*

1. *(Initialization Phase.) Key pairs $\{(PK_i, SK_i)\}_{i=1}^{n(k)}$ are generated by executing KeyGen$(1^k; \omega_i)$ for randomly chosen ω_i, and the set of public keys $S \doteq \{PK_i\}_{i=1}^{n(k)}$ is given to $\mathcal{A}$.*
2. *(Probing Phase I.) $\mathcal{A}$ is given access to a signing oracle $\mathcal{SO}(.,.,.)$, where $\mathcal{SO}(s, M, R)$ outputs $Sign_{s,SK_s}(M, R)$ and it is required that $R \subseteq S$ and $PK_s \in R$. $\mathcal{A}$ is also given access to a corruption oracle $\mathcal{CO}(.)$, where $\mathcal{CO}(i)$ outputs $\omega_i : 1 \leq i \leq n(k)$.*
3. *(Challenge Phase.) $\mathcal{A}$ outputs a message M, distinct indices i_0, i_1, and a ring $R \subseteq S$ for which $PK_{i0}, PK_{i1} \in R \cap S$ and all keys in R are distinct. If $(i_0, ., .)$ or $(i_1, ., .)$ was an input to SO, or if i_0 or i_1 was an input to $\mathcal{CO}$, $\mathcal{A}$ fails and the game terminates. Otherwise a random bit b is chosen, and $\mathcal{A}$ is given $\sigma \leftarrow Sign_{i_b,SK_{i_b}}(M, R)$.*
4. *(Probing Phase II.) $\mathcal{A}$ is again given access to SO and CO. If $(i_0, ., .)$ or $(i_1, ., .)$ is queried to SO, or if i_0 or i_1 is queried to $\mathcal{CO}$, $\mathcal{A}$ fails and the game terminates.*

5. *(Output Phase.) The adversary outputs a bit b', and succeeds if $b' = b$.*

Definition 11 *(**Linkable Anonymity w.r.t adversarially-chosen keys**). [4] A linkable ring signature is linkably anonymous w.r.t adversarially-chosen keys if for any PPT adversary $\mathcal{A}$ and for any polynomial n(.), the probability that $\mathcal{A}$ succeeds in the previous game, without restricting the ring R to be a subset of S in the challenge phase (but atleast two members of the ring belong to set S), and without requiring $R \subseteq S$ for $\mathcal{SO}(.,.,\mathcal{R})$ queries, is negligibly close to 1/2.*

Linkability

It is hard to generate the secret keys (e_1, e_2) for a given public key value x under the Strong RSA hardness assumption of *Accumulators with one-way domain*, hence the probability of producing a valid signature for a given transaction(message) and secret key pair is negligible. Security is ensured even in presence of adversarially-chosen keys.

Definition 12 *(**Linkability**).[4] A linkable ring signature is linkable if for any PPT adversary $\mathcal{A}$ and for any polynomial n(.), the probability that $\mathcal{A}$ succeeds in the following game is negligible :*

1. *(Initialization Phase.) As in Definition 9.*
2. *(Probing Phase.) As in Definition 9.*
3. *(Output Phase.) $\mathcal{A}$ outputs $(M_i^*, \sigma_i^*, R_i^*), i = 1, 2$, and succeeds if it holds that $\texttt{Verify}_{R_i^*}(M_i^*, \sigma_i^*) = 1$ and $R_i^* \subseteq S$ for $i = 1, 2$, $\texttt{Link}(\sigma_1^*, \sigma_2^*)$=0, and $|(R_1^* \cup R_2^*) \cap C| + |(R_1^* \cup R_2^*) \setminus S| \leq 1$, where C is the set of corrupted users.*

Definition 13 *(**Linkability w.r.t adversarially-chosen keys**).[4] A linkable ring signature is linkable w.r.t adversarially-chosen keys if for any PPT adversary $\mathcal{A}$ and for any polynomial n(.), the probability that $\mathcal{A}$ succeeds in the previous game, without restricting the rings R_0^*, R_1^* to be subsets of S (but atleast two members of each of the rings R_0^*, R_1^* belong to set S) in the output phase, and without requiring $R \subseteq S$ in $\mathcal{SO}(.,.,\mathcal{R})$ queries, is negligible.*

Non-slanderability

It ensures that no one can produce a linkable signature on behalf of or frame an honest signer ([39]). For our construction of FCsLRS, we consider a tag generation which provides *Transaction-Oriented* linkability. Since each tag generated is dependent on the transaction id and secret key pair and transaction id being unique for each transaction* and assuming the function $\tilde{H}$ is a random oracle , it is hard to produce two linked signatures for same transaction. Security is ensured even in presence of adversarially-chosen keys.

*transaction-id is the hash of the transaction payload

Definition 14 *(**Non-slanderability**).[4] A linkable ring signature is non-slanderable if for any PPT adversary $\mathcal{A}$ and for any polynomial n(.), the probability that $\mathcal{A}$ succeeds in the following game is negligible :*

1. *(Initialization Phase.) As in Definition 9.*
2. *(Probing Phase.) As in Definition 9.*
3. *(Output Phase.) $\mathcal{A}$ outputs $(\hat{\sigma}, M^*, \sigma^*, R^*)$ and succeeds if $R^* \subseteq S, \hat{\sigma}$ is the output of $\mathcal{SO}(\hat{s}, \hat{M}, \hat{R})$ for some $\hat{R} \subset S$ and $\hat{s}$ such that $PK_{\hat{s}} \in \hat{R} \cap S$, $\texttt{Verify}^*_R(M^*, \sigma^*)$=1, $\texttt{Link}(\hat{\sigma}, \sigma^*)$=1 and $\mathcal{A}$ never queried $\hat{s}$ to $\mathcal{CO}(.)$.*

Definition 15 *(**Non-slanderability w.r.t adversarially-chosen keys**).[4] A linkable ring signature is non-slanderable w.r.t adversarially-chosen keys if for any PPT adversary $\mathcal{A}$ and for any polynomial n(.), the probability that $\mathcal{A}$ succeeds in the above game, without the restrictions which the rings $\hat{R}, R^*$ are subsets of S (but atleast two members of each of the rings $\hat{R}, R^*$ belong to set S) in the output phase and $R \subseteq S$, in $\mathcal{SO}(.,.,\mathcal{R})$ queries is negligible.*

The following theorems [4] justify the security of our proposed construction under our proposed security model.

Theorem 3 *If the DDH in QR(N) problem, the LD-RSA problem, the Strong-RSA problem are hard and the function $\tilde{H}$ is random oracle, our construction is unforgeable.*

Proof. As discussed under **Security Notions**, if the scheme is *Non-Slanderable* and *Linkable* then it is implied to be *Unforgeable*. For if an adversary can forge a signature, he can either slander an honest user or collude with any user to break the linkability of the scheme. Therefore, the proof for is a direct consequence of Theorems 5 and 6.

Theorem 4 *Under the assumption that the threshold value t in the endorsement policy is greater than 1, if the DDH in QR(N) problem, the LD-RSA problem, the Strong-RSA problem are hard and the function $\tilde{H}$ is random oracle , then our construction is linkably-anonymous w.r.t. adversarially-chosen keys.*

Proof. Construct a simulator S from adversary $\mathcal{A}$, which wins game linkable-anonymity (as defined in *Definition* 10,11) with non-negligible advantage $1/2 + \epsilon$ over random guessing, to solve the LD-RSA problem under the DDH assumption.

S is given an instance of the LD-RSA problem $(n_0, n_1, T = g_{tid}^{p_b+q_b})$, where $g_tid = g^{\tilde{H}(tid)}$ (where $\tilde{H(.)}$ is a random oracle) , with $p_b.q_b = n_0$ or n_1 . S creates the system parameters correspondingly and randomly generate a set of key pairs $X = \{(PK_i, SK_i)\}$ using `KeyGen()`. It randomly chooses a bit $b' = 0$ or 1 and sets $PK^* = 2n_{b'} + 1$. Denote $X^* = X \cup \{PK^*\}$, which is then given to $\mathcal{A}$ as the set of public keys.

S handles the $\mathcal{SO}$ query as follows. For query involving $PK \in X$ as the signer, S is in possession of the secret key and can reply according to the algorithm specification.

There are two ways to handle the public key generated adversarially - first is to disallow the adversary from having this type of query(already knows the secret key) or the second way is that adversary has to prove the validity of the generated key pair to the simulator before it can be used. This models the scenario in practice when endorsers need to prove the validity of the key pairs before Fabric CA issues an enrolment certificate for it. In this case, the simulator extracts the secret key of the corresponding public key during the proof of validity of the key and the rest follows. Queries involving PK^* as the signer need special attention. S sets tag $\tilde{y} = T$ and computes the signature of knowledge. Under the DDH assumption in $QR(N)$, the simulated signature of knowledge (constructed as in Eq. 3.2.) is indistinguishable from the actual one if T is correctly formed. T is correctly formed if and only if $b = b'$.

If PK^* is chosen to be the challenge signature, then $\mathcal{A}$ wins the game with probability $1/2+\epsilon$, (ϵ being a negligible value), if T is correctly formed. On the other hand, $\mathcal{A}$ can only win with probability $1/2$ since the whole challenge signature is not related to either i_0 or i_1 .

In the challenge phase, with probability $2/|X^*|, PK^* \in \{i_0, i_1\}$. If T is correctly formed, then $\mathcal{A}$ wins the game with probability $1/2 + \epsilon$. Otherwise, $\mathcal{A}$ can only win with probability $1/2$. Thus, S solves the LD-RSA problem with probability $1/2+\frac{\epsilon}{2|X^*|}$ [†] which is non-negligible over random guessing.

Theorem 5 *If the DDH in QR(N) problem, the LD-RSA problem, the Strong-RSA problem are hard and the function $\tilde{H}$ is random oracle, then our construction is linkable w.r.t. adversarially-chosen keys.*

Proof Sketch. For adversary $\mathcal{A}$ to break the linkability property, it has to convince a verifier to accept a tag $\tilde{y}$ for which it cannot generate a correct signature of knowledge with any non-negligible probability. Then $\mathcal{A}$ must have conducted an incorrect proof in the signature of knowledge such that at least one of the following holds true: a successful forging of the constant-size ring signature ([13]) or incorrect proof of construction of $\tilde{y}$. The first part holds with negligible probability under the Strong-RSA assumption, and the second part is valid with negligible probability under the (computational) LD-RSA assumption. Thus, the total success probability is negligible.

Theorem 6 *If the DDH in QR(N) problem, the LD-RSA problem, the Strong-RSA problem are hard and the function $\tilde{H}$ is random oracle, then our construction is non-slanderable w.r.t. adversarially-chosen keys.*

Proof Sketch. Similar to the proof corresponding to 4. If adversary $\mathcal{A}$ is able to output a signature which slanders PK^* and since the soundness property of *Signature based*

[†] $Adv_{\mathcal{A}}^{LD-RSA} = Adv_{\mathcal{A}}^{Linkable-anonymity} \times Pr(\ selecting\ PK^*\ from\ X^*) \times Pr(b = b'|PK^*) = \epsilon \times \frac{1}{X^*} \times \frac{1}{2}$

on Proof of Knowledge (Eq. 3.2) holds, then there exists a *knowledge extractor (def. 3)* which can extract the secret key (p', q') corresponding to the public key PK^* such that $PK^* = 2p'q' + 1$. Hence simulator S solves the LD-RSA problem.

Chapter 5

Performance Analysis of FCsLRS *

The (`Sign,Verify`) algorithm pair just involves the execution of the Σ-protocol which is time independent from the number of public keys that were aggregated when constructing the accumulated value v and generation of witness value w. At the end of each protocol run, verifier $\mathcal{V}$ outputs a 0/1.

5.1 Theoretical Analysis

Table 5.1: Asymptotic complexity analysis of FCsLRS

Algorithm	Operations performed	Asymptotic complexity
Tag generation	$2E$	$\mathcal{O}(\lambda)$
Signature	$11E + 5M$	$\mathcal{O}(\lambda)$
Verification	$6E + 10M$	$\mathcal{O}(\lambda)$

E is an exponentiation (single base of the form g^a) operation, M is multibase exponentiation (of the form $g^a.h^b$) operation and λ is the security parameter.

5.2 Experimental Analysis

The performance of the signature scheme was measured on `Intel Core i5-4200U CPU, quad core processor, frequency 1.60 GHz`, OS : *Ubuntu-16.04 LTS* (64 bit). The programming language used is `Go 1.10`, packages used is *crypto, golang.org/x/crypto/sha3, rand* and *math*. The code for *Cyclic Group Generator* is based on the one given under `Project-iris`[†]. For the ananlysis of the signature generation and verification time, the RSA modulus size, also the security parameter λ, has been

*Over here we analyze the performance for generation and verification of one signature

[†]https://github.com/project-iris/iris/blob/v0.3.2/crypto/cyclic/cyclic.go

varied as 1024 bits, 2048 bits and 3072 bits. The endorsement set size (number of participants, n) was varied as $4, 8, \ldots, 256$, in ascending powers of 2 and message length was varied as 2KB,4KB and 8KB respectively. The functions H_1 and $\tilde{H}$ used is *SHA-3* producing a message digest of size 128 bits. The range of both the hash functions must be a subset of $QR(N), N$ is the RSA-modulus. Also as per Eq (4), any element in the group $QR(N)$ when raised to the power of a number in the group $\mathbb{Z}_{N/4}$, again returns an element which belongs to $QR(N)$. Since N is varied between 1024 bits to 3072 bits and least value of $N/4$ is 256 bits, so a message digest of 128 bit is definitely falling in the range $\mathbb{Z}_{N/4}$ for any case. We had to do this approximation since selection of an element from $QR(N)$ when $\phi(N)$ is unknown is hard.

10 instances for each message length was generated and in total, the FCsLRS algorithm was executed for 630 instances. However the execution time remains invariant of change in message length for a given number of participants and RSA modulus size . Thus for ease of tabulation, average over all 30 instances of such values was taken. Time taken by the signature generation and verification algorithm with respect to number of endorsers has been recorded in Table 5.2 and Table 5.3. Corresponding graphs has been plotted in Fig.5.1 and Fig. 5.2 respectively. The signature generation time and verification time remains constant for a fixed value of RSA modulus size. On varying the λ value, increase in computation time has been observed for both the algorithms. The Golang code for signature generation of *1-out-of-n* endorsement policy is available in [2]. Hence not only the signature size, but also the execution time of the code remains constant for a given value of λ.

Table 5.2: Signature generation time for FCsLRS vs number of participants

Endorsement set size	Signature generation time(ms) for 1024 bit RSA modulus	Signature generation time(ms) for 2048 bit RSA modulus	Signature generation time(ms) for 3072 bit RSA modulus
4	26.3810697	153.494131	484.8454402
8	25.9974438	156.6600916	467.8548517
16	25.9047032	153.3631163	462.5272153
32	26.0976248	153.1091731	467.6555261
64	25.9170626	153.0620514	482.5397836
128	25.187382	152.0527724	462.7118533
256	25.6127477	152.1151231	462.6459938

Table 5.3: Verification time for FCsLRS vs number of participants

Endorsement set size	Verification time(ms) for 1024 bit RSA modulus	Verification time(ms) for 2048 bit RSA modulus	Verification time(ms) for 3072 bit RSA modulus
4	33.8444186	189.9202852	586.5080154
8	34.1522705	194.7279413	574.2996354
16	33.7478959	190.1404563	567.9998257
32	33.8859284	190.7751803	577.9523215
64	34.4397107	190.8586968	580.6172581
128	34.1952231	191.0990971	569.9051838
256	34.3507846	191.6187186	564.1658533

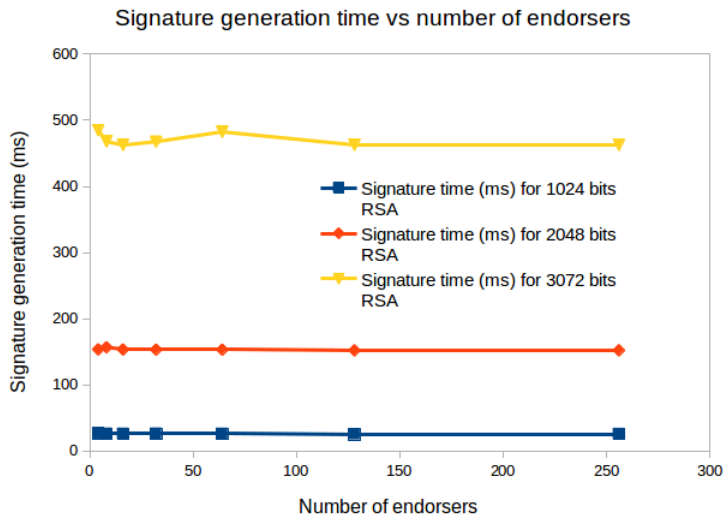


Figure 5.1: Signature Run time vs endorsement set size plot

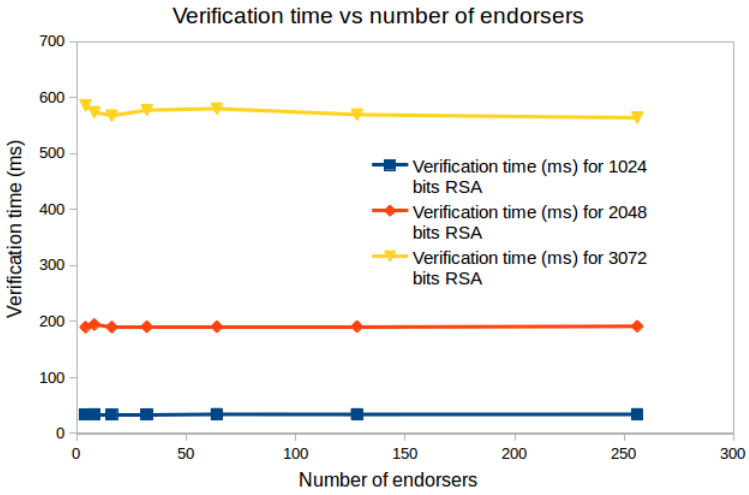


Figure 5.2: Verification Run time vs endorsement set size plot

5.3 Description of the Implementation

In this section, we give a high level description of the main methods for *1-out-of-n* endorsement policy. Assuming that a signer S wants to endorse a transaction with transaction id as $tid = hash(transaction\ payload)$ and transaction payload denoted by $m \in \mathcal{M}$, where $\mathcal{M} \in 0, 1^*, S \in \mathcal{E}$, $\mathcal{E}$ is the endorsement set, secret key is $sk_S = (p_\pi, q_\pi)$ and public key of S is $pk_S = 2p_\pi.q_\pi + 1$. Since we consider the case of just one endorser, we eliminate the code for check of linkability match as of now. But when it is integrated, check for linkability must be added for each transaction.

1. **Initialization.** This step involves generation of the RSA Modulus integer N of size λ bits. This step is executed by Fabric CA which generates the values by taking the security parameters as its input. To find a generator of $QR(N)$, we use the following lemma ([30]) :

   Lemma 1 *Let $N = p.q$ be the product of two distinct safe primes, and $u \in QR(N)$ a quadratic residue. Then u is a generator for $QR(N)$ if and only if $gcd(u - 1, N) = 1$.*

Procedure 1: Initializations

Input : λ

Output: Public parameters : N, g, h, t, y, s, ζ

1. Generate 2 prime $p, q, p \neq q$.
2. Generate 2 safe primes $p, q : p = 2p' + 1, q = 2q' + 1, |p| = |q| = \frac{\lambda}{2}$.
3. Find $N = p.q$.
4. Find a generator of the group $QR(N)$ using Lemma 1. Let that be u.
5. u generates g, h, t, y, s, ζ using some random discrete logarithm value $rd_i, 1 \leq i \leq 6, 2 \leq rd_i \leq |QR(N)| - 1$ where $|QR(N)| = p'.q'$.

---

2. **Key Generation.** Given an input n, which is the number of endorsers, each of the endorsers generate their own public key and private key pairs independently. (It only proves using zero-knowledge to Fabric CA about the correctness of the public key generated [‡]). Upon key generation , these values are made available in the public database $\mathcal{DB}$. The procedure mentioned below must be run parallely for each endorser present in endorsement set $\mathcal{E}$.

[‡]In our implementation, since we have developed the signature scheme as an independent module without considering any Public Key Infrastructure, so for the ease of implementation we have assumed the public keys generated by each endorser is correct.

Procedure 2: Key Generation for endorser E_i

Input : $\lambda, l, \mu : \lambda > l - 2, \frac{l}{2} > \mu + 1$, where $\mathcal{E}$ is the endorsement set
Output: Public Key : pk_i, Secret Key : sk_i

1. Generate 2 prime $p, q, p \neq q : q \in (2^{\frac{l}{2}} - 2^{\mu} + 1, 2^{\frac{l}{2}} + 2^{\mu} - 1)$. $sk_i = (p, q)$.
2. Generate $pk_i : pk_i = 2p.q + 1$.
3. Send pk_i to database $\mathcal{DB}$.

3. **Public Key accumulation**. Fabric CA uses its accumulator with one-way domain to generate an accumulated value of all the public keys in $\mathcal{DB}$, each having valid enrolment certificate.

Procedure 3: Accumulated value computation

Input : all pk's in database $\mathcal{DB}$, generator $u, \langle u \rangle = QR(N)$
Output: Accumulated value : v

```
v ← u
for pk_i ∈ DB do
    v ← v^{pk_i} mod N
end
```

4. **Witness Generation for Signer S**. Signer $\mathcal{S}$ can generate the witness w using values of all public keys forming the ring except its own public key.

Procedure 4: Witness value for signer S

Input : all pk's in database $\mathcal{DB}$, generator $u, \langle u \rangle = QR(N)$, Signer S public key : pk_S
Output: witness value : w

```
w ← u
for pk_i ∈ DB : pk_i ≠ pk_S do
    w ← w^{pk_i} mod N
end
```

5. **Tag Generation**. To generate the tag, the signer S needs to compute $g_t id$ from g given the transaction id tid.

6. **Computation of public values for Signature based on Proof of Knowledge Construction**. Signer S computes public values T_1, T_2, T_3, T_4, T_5 where $T_1 = g_{tid}^r \mod N, T_2 = (h^r \zeta^{pk_S + r}) \mod N, T_3 = (s^r g_{tid}^{q_\pi}) \mod N, T_4 = (w.y^r) \mod N, T_5 = (t^r g_{tid}^{2p_\pi}) \mod N$.

Procedure 5: Tag generation for signer S

Input : Signer S secret key : $sk_S = (p_\pi, q_\pi)$, transaction id : tid
Output: tag value : $\tilde{y}$

1 $x \leftarrow \tilde{H}(tid)$
2 $g_{tid} \leftarrow g^x \mod N$
3 $\tilde{y} \leftarrow g_{tid}^{p_\pi + q_\pi} \mod N$

Procedure 6: Public value generation by signer S

Input : Signer S secret key : $sk_S = (p_\pi, q_\pi)$, public key pk_S, g_tid, witness value : w
Output: Public values : T_1, T_2, T_3, T_4, T_5

1 $r \xleftarrow{R} \mathbb{Z}_{N/4}$
2 Compute T_1, T_2, T_3, T_4, T_5 as per equations mentioned above.

7. **Signature Generation**. Signer generates the challenge value which can be generated again at the verifier side as well. This is the standard *Fiat-Shamir Transformation* which has been used. Send all these value (mentioned in the output of *Signature Algorithm* along with tag $\tilde{y}$ to verifier $v \in \mathcal{V}$.

Procedure 7: Signature

Input : r,Signer S secret key : $sk_S = (p_\pi, q_\pi)$, public key pk_S, message $m \in \mathcal{M}$
Output: $u_1, u_2, \ldots, u_9, \tilde{\alpha_1}, \tilde{\alpha_2}, \ldots, \tilde{\alpha_5}$

1. Generate $\alpha_i, 1 \leq i \leq 3 : 0 < \alpha_i < N/4 - 1$.
2. Compute $u_1, u_2, \ldots, u_9$ as per Eq. 3.4.
3. Computes the challenge value $c = H_1(m, u_1, u_2, \ldots, u_9)$, where H_1 is a random oracle.
4. Using c, compute $\tilde{\alpha_1}, \ldots, \tilde{\alpha_5}$ as per Eq. 3.5.

8. **Verification Algorithm**. Verifier v computes c using the values sent to it by signer S. Using equations under Eq. 3.6 it's going to check whether the *Signature based on Proof of Knowledge* construction is correct or not.

The main methods described in section 5.3 is illustrated here.

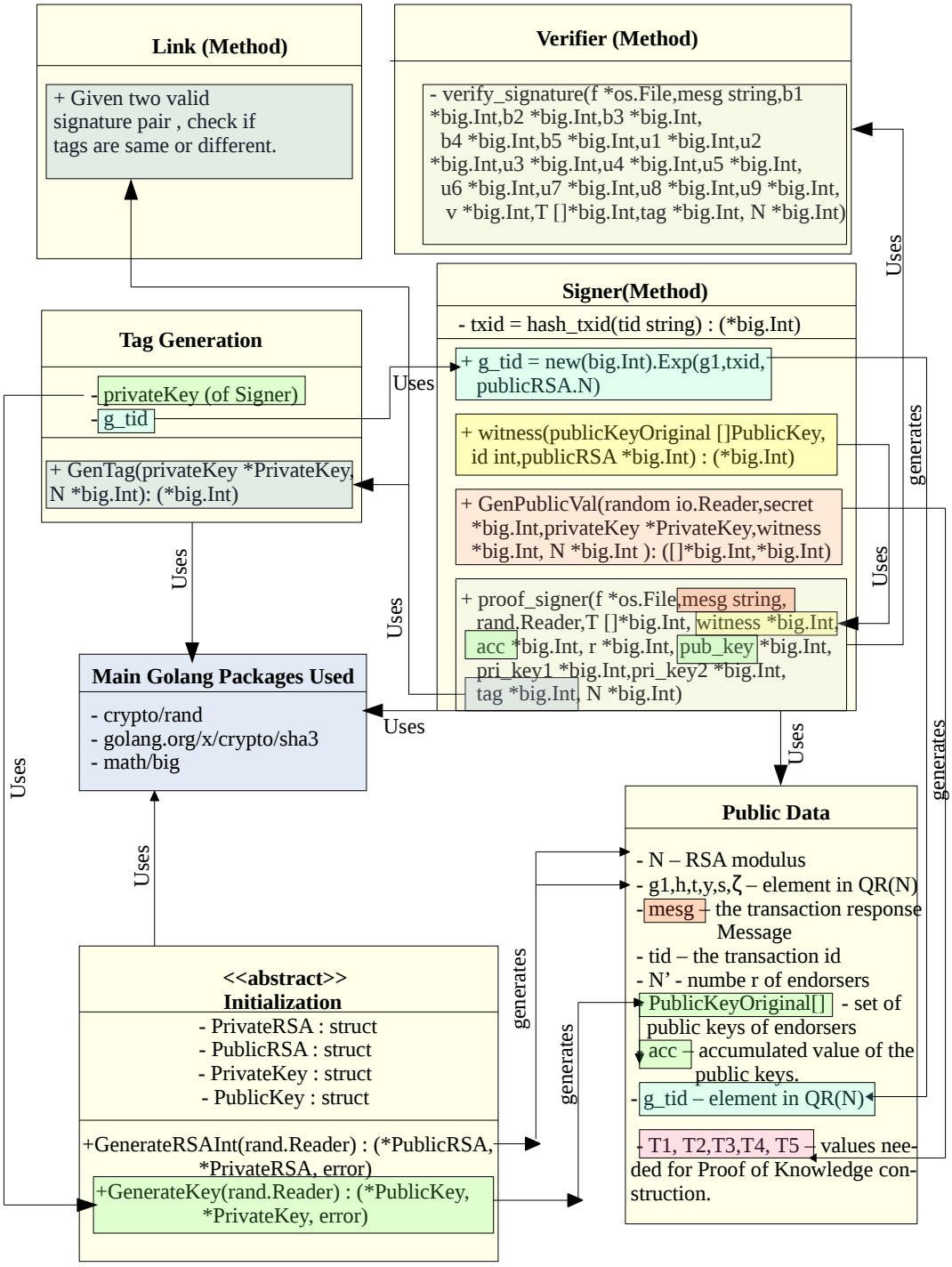


Figure 5.3: Class Diagram

Chapter 6

Integration of FCsLRS Module in Hyperledger Fabric

In current workflow of transaction ([23]), to invoke a transaction, the client sends a "`PROPOSE` message" to a set of endorsing peers of its choice. The set of endorsing peers for a given *chaincodeID* is made available to client via peer, which in turn knows the set of endorsing peers from endorsement policy. The format of a `PROPOSE message` is ⟨*PROPOSE,tx,[anchor]*⟩ as shown in Fig. 6.1.

On reception of a `PROPOSE message` message from a client, the endorsing peer *epID* first verifies the client's signature *clientSig* and executes the transaction (txPayload). This peer node forwards internally `tran-proposal` (and possibly transaction) to the part of its (peer's) logic that endorses a transaction, referred to as endorsing logic. By default, endorsing logic at a peer accepts the `tran-proposal` and simply signs the `tran-proposal`. However, endorsing logic may interpret arbitrary functionality, for e.g., interact with legacy systems with `tran-proposal` and transaction as inputs, to reach the decision whether to endorse a transaction or not. If endorsing logic decides to endorse a transaction, it sends a `PROPOSE-RESPONSE` packet containing the message :⟨*TRANSACTION-ENDORSED, tid,* ***tran-proposal****,epSig*⟩ message (as shown in Fig. 6.2) to the submitting client, where: `tran-proposal` := *(epID,tid,chaincodeID,txContentBlob, readset,writeset)*, where txContentBlob is chaincode/transaction specific information. The intention is to have txContentBlob used as some representation of transaction (e.g., txContentBlob=tx.txPayload). *epSig* is the endorsing peer's signature on `tran-proposal`. Else, in case the endorsing logic refuses to endorse the transaction, an endorser may send a message ⟨*TRANSACTION-INVALID, tid, REJECTED*⟩ to the submitting client.

To integrate the *Constant-Sized linkable ring signature* module, we first have to change the `PROPOSAL RESPONSE` format (as shown in Fig. 6.3). In the source code, under *hyperledger/fabric/protos/peer/proposal_response.proto* in the structure `ProposalResponse`, add a field called as **Tag** which will enable *Transaction-oriented* linkability. The structure `Endorsement` must be changed by deletion of the field

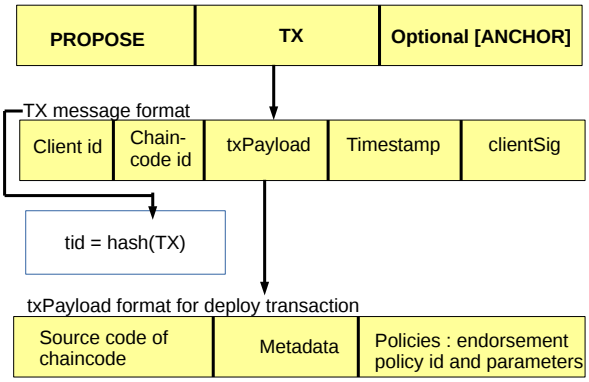


Figure 6.1: PROPOSE message format in Hyperledger Fabric

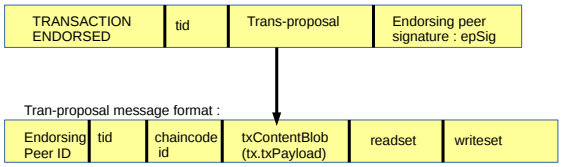


Figure 6.2: PROPOSE RESPONSE message format in Hyperledger Fabric

endorser (data type bytes[ ]) which reveals the endorsing peer ID. Create a `FCsLRS`* package under *hyperledger/fabric/bccsp* which can be used by the signer to sign the message. In the file *hyperledger/fabric/msp/identities.go*, delete the field **identity** in the structure `signingidentity`. The `Verify` function will just check the validity of the signature corresponding to a message. Instead of checking the identity of signer, it will verify whether signer can give a *Signature based on Proof of Knowledge* of the secret to prove its membership to the *Endorsement set* $\mathcal{E}$.

The *Transaction flow diagram* for *1-out-of-n* endorsement policy is given in Fig.

*Fabric's Constant-Sized Linkable Ring Signature

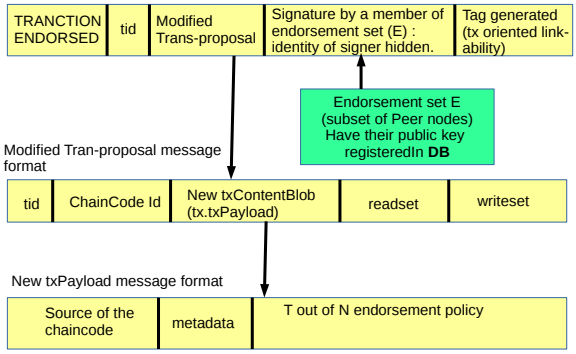


Figure 6.3: Modified PROPOSE RESPONSE message format in Hyperledger Fabric

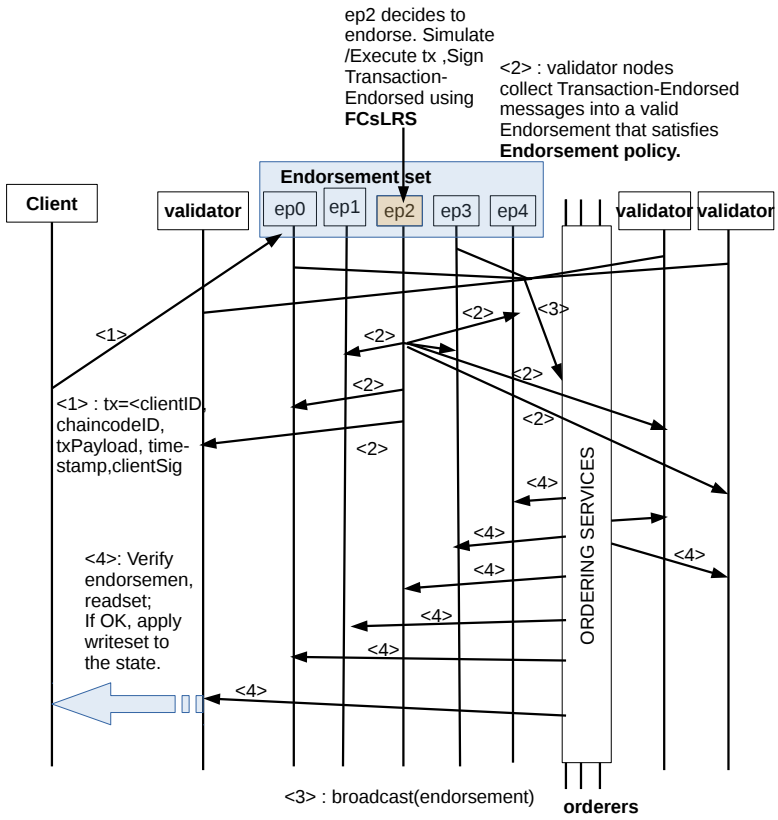


Figure 6.4: Modified transaction flow diagram when any one endorser is willing to endorse

6.4. It is clear that when *endorsing peer ep2* decides to endorse the transaction, it just forms a ring with rest of endorser in the *Endorsement set* to construct the ring signature and the `PROPOSAL RESPONSE` format being same as that shown in Fig.6.3. This is broadcasted to all the *Peer* nodes ($\langle 2 \rangle$ of Fig. 6.4). If majority of the nodes reach a consensus on receipt of a valid endorsement, then one of the *validator* node forwards it to the *Ordering Service*($\langle 3 \rangle$ of Fig. 6.4). From this step onwards, the transaction flow is same as that shown in Fig. 2.1.

Chapter 7

Conclusion and Future Work

We have discussed about the design of an anonymous endorsement system for Hyperledger Fabric and gave the construction of a new constant sized linkable ring signature scheme, FCsLRS. We plan to incorporate this module in future release of Fabric and evaluate the performance in the actual setting. In our future work, we aim to provide a construction of short ring signature scheme, probably using pairing based cryptography.

Currently, verifiers are required to count individual valid ring signature and check if the aggregate is above the threshold in order to implement threshold endorsement policy. We would like to replace it with threshold signature scheme which can guarantee the same or better level of anonymity as it is offered now by the proposed system.

Since the endorsement policy is specific to Hyperledger Fabric, we would like to explore other permissioned blockchain systems and check whether the proposed scheme can be extended for other use cases.

Bibliography

[1] What are zk-snarks? https://z.cash/technology/zksnarks.html (2017)

[2] Golang code : Fcslrs. https://www.dropbox.com/sh/8zr0yvzuimtnyuu/AACUk0z8qRHCwecxXEIfePJ6a?dl=0 (2018)

[3] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al.: Hyperledger fabric: A distributed operating system for permissioned blockchains. arXiv preprint arXiv:1801.10228 (2018)

[4] Au, M.H., Chow, S.S., Susilo, W., Tsang, P.P.: Short linkable ring signatures revisited. In: European Public Key Infrastructure Workshop. pp. 101–115. Springer (2006)

[5] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM conference on Computer and communications security. pp. 62–73. ACM (1993)

[6] Bresson, E., Stern, J., Szydlo, M.: Threshold ring signatures and applications to ad-hoc groups. In: Proceedings of the 22Nd Annual International Cryptology Conference on Advances in Cryptology. pp. 465–480. CRYPTO '02, Springer-Verlag, London, UK, UK (2002)

[7] Camenisch, J., Michels, M.: Proving in zero-knowledge that a number is the product of two safe primes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 107–122. Springer (1999)

[8] Camenisch, J., Michels, M., et al.: Separability and efficiency for generic group signature schemes. In: Annual International Cryptology Conference. pp. 413–430. Springer (1999)

[9] Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: Annual International Cryptology Conference. pp. 410–424. Springer (1997)

[10] Camenisch, J., Van Herreweghen, E.: Design and implementation of the idemix anonymous credential system. In: Proceedings of the 9th ACM conference on Computer and communications security. pp. 21–30. ACM (2002)

[11] Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Annual International Cryptology Conference. pp. 78–96. Springer (2006)

[12] Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Advances in Cryptology—CRYPTO'94. pp. 174–187. Springer (1994)

[13] Dodis, Y., Kiayias, A., Nicolosi, A., Shoup, V.: Anonymous identification in ad hoc groups(full version). In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 609–626. Springer (2004)

[14] Dreifuerst, D.: Permissioned vs. permissionless blockchains: Who will win and will it matter? `https://medium.com/dustindreifuerst/permissioned-vs-permissionless-blockchains-acb8661ee095` (2018)

[15] Feige, U., Fiat, A., Shamir, A.: Zero-knowledge proofs of identity. Journal of cryptology 1(2), 77–94 (1988)

[16] Franklin, M.K., Zhang, H.: A framework for unique ring signatures. IACR Cryptology ePrint Archive 2012, 577 (2012)

[17] Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In: International Conference on Applied Cryptography and Network Security. pp. 156–174. Springer (2016)

[18] Goldfeder, S., Gennaro, R., Kalodner, H., Bonneau, J., Kroll, J.A., Felten, E.W., Narayanan, A.: Securing bitcoin wallets via a new dsa/ecdsa threshold signature scheme (2015)

[19] Goldreich, O.: Foundations of Cryptography: Volume 1. Cambridge University Press, New York, NY, USA (2006)

[20] Hardjono, T., Smith, N., Pentland, A.: Anonymous identities for permissioned blockchains. Technical report (2014)

[21] Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification. Tech. rep., Tech. rep. 2016-1.10. Zerocoin Electric Coin Company (2016)

[22] IBM, L.F..: hyperledger-fabric-ca Documentation , Release master. Read the Docs (2018)

[23] IBM, L.F..: hyperledger-fabricdocs Documentation , Release master. Read the Docs (2018)

[24] Kadiyala, A.: Nuances between permissionless and permissioned blockchains. https://medium.com/akadiyala/nuances-between-permissionless-and-permissioned-blockchains-f5b566f5d483/ (2018)

[25] Kate, A., Goldberg, I.: Distributed private-key generators for identity-based cryptography. In: International Conference on Security and Cryptography for Networks. pp. 436–453. Springer (2010)

[26] Kiayias, A., Tsiounis, Y., Yung, M.: Traceable signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 571–589. Springer (2004)

[27] Kogias, E.K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 279–296 (2016)

[28] Laurence, T.: Blockchain for Dummies. John Wiley & Sons (2017)

[29] Mercer, R.: Privacy on the blockchain: Unique ring signatures. arXiv preprint arXiv:1612.01188 (2016)

[30] Micciancio, D.: The rsa group is pseudo-free. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 387–403. Springer (2005)

[31] Narula, N., Vasquez, W., Virza, M.: zkledger: Privacy-preserving auditing for distributed ledgers. auditing 17(34), 42 (2017)

[32] Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 387–398. Springer (1996)

[33] Rivest, R., Shamir, A., Tauman, Y.: How to leak a secret. Advances in Cryptology—ASIACRYPT 2001 pp. 552–565 (2001)

[34] Stathakopoulou, C., Cachin, C.: Threshold signatures for blockchain systems. https://domino.research.ibm.com/library/cyberdig.nsf/papers/CA80E201DE9C8A0A852580FA004D412F/$File/rz3910.pdf (2017)

[35] Sun, S.F., Au, M.H., Liu, J.K., Yuen, T.H.: Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In: European Symposium on Research in Computer Security. pp. 456–474. Springer (2017)

[36] Tsang, P.P., Wei, V.K.: Short linkable ring signatures for e-voting, e-cash and attestation. In: ISPEC. vol. 3439, pp. 48–60. Springer (2005)

[37] Tsang, P.P., Wei, V.K., Chan, T.K., Au, M.H., Liu, J.K., Wong, D.S.: Separable linkable threshold ring signatures. In: Indocrypt. vol. 3348, pp. 384–398. Springer (2004)

[38] Van Saberhagen, N.: Cryptonote v 2. 0 (2013)

[39] Wei, V.K.: Tracing-by-linking group signatures. In: International Conference on Information Security. pp. 149–163. Springer (2005)

[40] Yuen, T.H., Liu, J.K., Au, M.H., Susilo, W., Zhou, J.: Efficient linkable and/or threshold ring signature without random oracles. The Computer Journal 56(4), 407–421 (2012)