CLOAKING FABRIC, A CONFIDENTIALITY LAYER FOR HYPERLEDGER FABRIC

BY

RAHUL SHIVU MAHADEV

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Assistant Professor Andrew E Miller

ABSTRACT

Permissioned blockchains have resulted in some unlikely collaborations between organizations that would have previously been impossible due them mutually distrusting each other. They provide a sense of trust among the parties due to the decentralized nature of their deployment that prevents censorship from a subset of the parties. Decentralization mandates that all the parties have the same view of the system, therefore it has been difficult to represent and store private data. Asynchronous Verifiable Secret Sharing(AVSS) and Secure Multi Party Computation(MPC) are techniques from cryptography that allow the sharing of secrets among multiple parties and enable arbitrary computations on the shared data without leaking any information about the data.

Previously, AVSS and MPC protocols were inefficient for practical use or did not work in the same setting of blockchains where nodes of the blockchain could arbitrarily fail. Honeybadger AVSS and Honeybadger MPC are robust and scalable frameworks that make them a good candidate to be coupled with a permissioned blockchain to form a confidentiality layer on top of it. We present Cloaking Fabric, an extension to the popular permissioned blockchain Hyperledger Fabric that utilizes HoneybadgerMPC and HoneybadgerAVSS to provide a confidentiality layer that would allow smart-contracts on the blockchain to interact with private data. We present a suite of applications to demonstrate our system and measure the overhead it would have over standard MPC operations.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I'd like to acknowledge everyone at Decentralized Systems Lab at the University of Illinois, Urbana-Champaign for all the support and help during my thesis. I'd also like to thank my friend circle of Kartik, Sathwik and Vandana for standing by me. Special thanks to Rahul Govind for Fixed Point stuff and being a voice of reason throughout.

I would like to thank my Adviser Professor Andrew Miller and Dr. Angelo De Caro from IBM Research for constant feedback and suggestions during the course of the project.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Organizations typically maintain and manage their own database for their operations[1][2]. There could be scenarios where an organization interacts with other organizations working with them. Consider a supply chain setting[3] where there are usually many intermediate organizations involved between the source and the final consumer and each of them share some data points with others. Traditionally the interaction between organizations that requires data sharing is handled by a trusted third party or by one of the organizations themselves.

Aggregating petabytes of data under the control of one organization could have catastrophic impact. There have been a incidents like the Cambridge Analytica incident[4] where collected user data was sold and wrongfully misused to direct targeted advertisements to Facebook users. Some[5, 6] say that this incident had an impact on the 2016 US Presidential Elections. This has highlighted the intensity of impact with regard to misuse of data. Laws like the GDPR(General Data Protection Regulation)[7] act by the European Union have been since passed which aim to enforce strong constraints to how data is stored and used[8] . This makes it important to have secure and scalable methods to allow collaboration between organizations without revealing part or entirety of one party's data to another.

A new variant of distributed systems called "blockchains" have emerged in the recent years. Blockchains are replicated append-only databases where every participant maintains a copy of the ledger and can participate in the consensus for deciding the new transactions. Additionally blockchains can allow the execution of arbitrary programs called smart-contracts. Blockchains have soared in popularity since the introduction of Bitcoin a peer-to-peer electronic currency[9] in 2009. There are numerous applications running on blockchains as they are resistant to censorship due to a virtue of resilience by a replicated ledger. The CEO of IBM has described blockchains to be the next big technology and predicts it to have the same impact to what the Internet had for communication but with trusted transactions. [10]. Blockchains are being used among companies in areas of health care[11, 12, 13] and finance[14] to share data and collaborate.

Hyperledger Fabric[15] is a blockchain for the permissioned setting which caters to scenarios requiring better performance and throughput compared to that of permissionless blockchains found in cryptocurrencies like Bitcoin and the permissioned nature makes it easier to manage identities. This makes it ideal for use among organizations striving for a

similar goal but are distinct entities and mutually distrust each other. However, blockchains rarely provide mechanisms to support storing of confidential data and computing functions without revealing the data. In the most common deployment scenario Fabric would work as a transparent, fault tolerant and censorship resistant ledger so that all parties can maintain the data at all times so that a subset of them cannot delete or destroy the data. This makes the joint operations between organizations more meaningful. However, there are various scenarios where a portion of the data on the blockchain needs to be private due to restrictions and laws set by governing bodies[16]. Blockchain deployments deployments[11, 13, 12] overcome HIPAA[16] restrictions by making the restricted data freely available but log the usage of data so that unauthorized access can be enforced later. Some scenarios where private data is useful on blockchains are as follows:

- An organization might have setup complex operations pipelines involving some of its private data as well. The application would be written for the blockchain and the private data would need to be used by the application. For example, consider a hospital which shares a blockchain with the pharmaceutical company. The hospital could have setup orders and billing on the blockchain, the hospital would identify the orders with patient information, it would be convenient if confidential patient information could be made private and only the type and quantity of the order be made public.

- There are scenarios[17][18] where data held by one organization is not sufficient to analyze meaningful trends or patterns. Multiple organizations need to perform data mining together on their confidential data. This is especially true for hospitals and financial firms where data is private. For example, it is in the best interest of hospitals to have higher success rates on diagnosing, treating and controlling ailments and diseases. But often a single hospital may not have enough information to conclude about the best practices. Sharing their data with other hospitals is not straightforward as there are privacy laws which restrict the usage of patient data.

Another burgeoning technology is Secure Multi Party Computation(MPC), In MPC the goal is to allow multiple parties to jointly compute a function over every party's inputs while keeping the inputs private. MPC can be used to replace a trusted third party in a variety of scenarios. Storing secret data in a blockchain with efficient access control is a non-trivial problem to solve as the principle of blockchains requires the data to be replicated among all the parties. Existing solutions[19, 20] for private data fall back to centralized components in their protocol or avoid sharing the data itself but share a key among parties and store the same encrypted data on all nodes(physical instance of a party) of the blockchain.

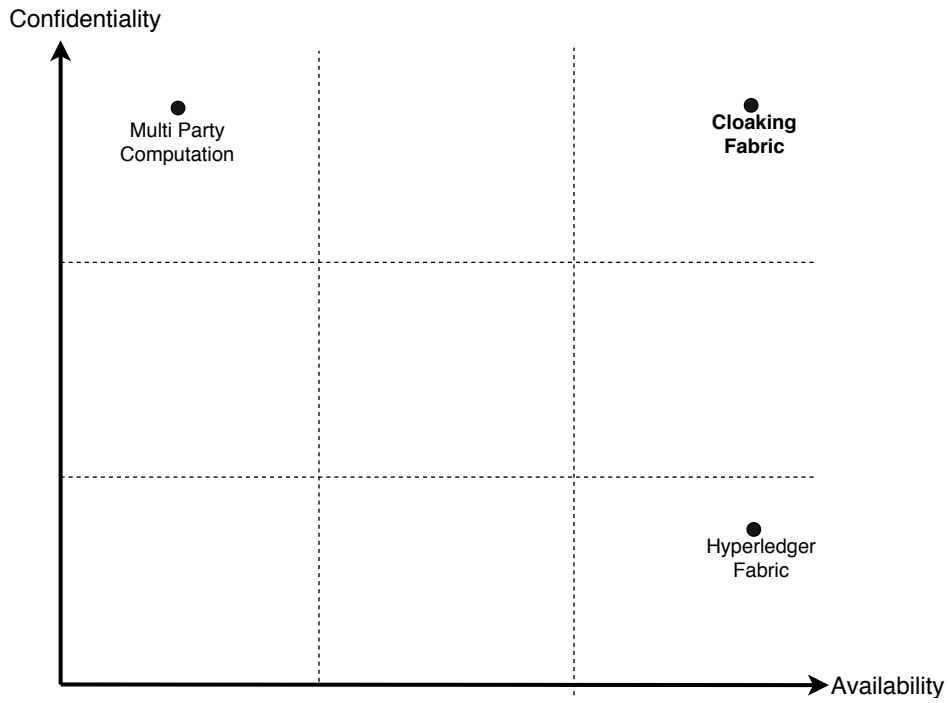

Figure 1.1: Trade off between Availability and Confidentiality

It seems natural that coupling MPC with blockchains would give us a valuable symbiosis between them that results in confidentiality and availability of data. In this thesis we present Cloaking Fabric, a confidentiality layer for Hyperledger Fabric which aims to manage secret data on the blockchain and allow arbitrary computations on them by providing a secret namespace for chaincode(smart-contracts) to use. Figure 1.1 is an illustration of the properties of blockchains vs properties of MPC toolkits, Multi-Party Computation provides confidentiality but lacks in availability compared to blockchains. Cloaking Fabric would have properties of both blockchains and MPC systems hence it is plotted on the far right corner of the graph.

It has been challenging for such a system to exist in the past due to the following reasons.

- Permissioned blockchains were not customizable enough to build a confidentiality layer on top of it efficiently.
- MPC toolkits in the past did not scale well with a large number of nodes and were usually not fault-tolerant.

For Cloaking Fabric, we present a confidentiality layer for Hyperledger Fabric built using HoneybadgerMPC. HoneybadgerMPC presents a framework of protocols (Honeybadger AVSS, Batch reconstruction, etc.) that work in a robust setting similar to blockchains where

a threshold number nodes can arbitrarily fail.

For Cloaking Fabric, we utilize Hyperledger Fabric and HoneybadgerMPC to build a system capable of availability and confidentiality. A node, peer or party in Fabric would be equivalent to a party in HoneybadgerMPC. Hyperledger Fabric is a widely deployed permissioned blockchain[21] that provides a strong base of features and supports customizability. HoneyBadgerMPC [22] is a robust(can tolerate a certain number arbitrary node failures) MPC framework.

To summarize, this thesis will present

1. **Cloaking Fabric**, a Confidentiality Layer for Hyperledger Fabric which enables chaincodes to interact with private data and perform arbitrary computations with them.

2. An API for building applications which interact with private data. The API extends on the chaincode programming model and adds suitable features to easily build applications.

3. A suite of applications to demonstrate the capabilities of Cloaking Fabric. We benchmark the running times of these applications and compute the overhead of using a blockchain along with a MPC framework.

CHAPTER 2: BACKGROUND

This chapter explains the relevant background required to understand the remainder of the thesis. Section 2.1 introduces the concept of blockchains and their different flavors with regard to setting they are deployed in. Section 2.2 gives an overview into Hyperledger Fabric and details the individual components that are used in Cloaking Fabric. Section 2.3 explains Secret Sharing and Section 2.4 explains Secure Multi Party Computation.

2.1 BLOCKCHAIN

A blockchain is an immutable distributed ledger shared between mutually distrusting parties where every party maintains a copy of ledger state. They were first made popular by Bitcoin[9] which is a peer-to-peer electronic cash system(cryptocurrency).
The nodes execute a consensus protocol to order and validate transactions and maintain consistency among themselves. There are two kinds of blockchain systems based on membership.

- Permissionless blockchains: In this type of blockchain anyone can download the node software and participate without a specific identity. Some examples of permissionless blockchains are Bitcoin and Ethereum. It is likely that a permissionless blockchain is often coupled with a cryptocurrency to provide a economic incentive for the nodes to participate in consensus.

- Permissioned blockchain: Permissioned blockchains, on the other hand, restrict membership to parties which can be identified and are known to each other. Permissioned blockchains work best in a consortium setting, when the parties know each other and have a common goal but are mutually distrustful of others e.g banks which transact with each other, organizations in the same supply chain.

Some blockchains also allow the execution of arbitrary logic, first introduced as scripts in Bitcoin and later as Smart Contracts in Ethereum[23]. Smart Contracts are popular for building trusted distributed applications owing to the decentralized nature of blockchains. Smart Contracts are usually required to be written in a domain specific programming language as in the case of Solidity for Ethereum.

2.2 HYPERLEDGER FABRIC

Hyperledger Fabric is an open source blockchain project started by IBM and now hosted by the Linux Software Foundation. It aims to be permissioned blockchain with an emphasis on modularity where components of the blockchain such as consensus module can be easily swapped according to requirement.

Fabric aims to solve the following problems of prior blockchains:

- Fabric supports a per-application trust model with the help of endorsement policies that specify the endorsement criteria for committing a transaction.
- Pluggable consensus modules in fabric help to satisfy the sentiment that there is no "one size fits all" consensus protocol[24]. This allows Fabric to be deployed in differing settings.
- Chaincodes in Fabric can be developed using general purpose programming languages like Go and NodeJS. This sets a low barrier to entry for application developers and allows for more complex applications due to the enormous support available in the form of libraries.

2.2.1 Architecture of Fabric

Fabric overcomes the above limitations and aims to be a blockchain that focuses on resiliency, flexibility, scalability and confidentiality.

2.2.2 Order Execute Architecture

Blockchains follow the blueprint of State-Machine-Replication(SMR) which has been studied extensively in Distributed Systems literature[25][26]. Most traditional blockchain systems follow the "order-execute" architecture. What this means is that the blockchain orders the transactions first and runs them on all the nodes via State Machine Replication.

Typically, consensus in blockchains work as follows

1. Every participating node would create a block with all valid transactions.
2. The node would try and solve a cryptographic puzzle(Proof of Work).

3. If the node was able to solve the puzzle it will disseminate via a gossip protocol the block along with the solution to the puzzle.

4. On receiving the block every node executes the transactions in the block **in the same order** and validates it.

In popular permissioned blockchains such as Tendermint[27], Chain[28] or Quorum[29] some variant of Byzantine Fault Tolerant consensus protocol is used. A popular variant is PBFT[30]. They perform state machine replication by ordering the transactions first and then executing the transactions in order.

Hyperledger Fabric is an exception to this, it drops Order-Execute for an **Execute and then Order** architecture which allow it to achieve better throughput, scalability and flexibility in writing smart contracts.

This can be described in three steps

1. Executing a transaction and checking its correctness.

2. Ordering through a consensus protocol irrespective of transaction semantics.

3. Validating the transaction according to the per-application trust assumptions.

This deviates from Order-Execute paradigm discussed in the previous section in the sense that Fabric executes transactions before reaching an agreement on their order by combining two well-known approaches to replication.

1. Passive replication: Similar to replication in distributed databases, here every transaction is executed on multiple peers. This can be thought of as pre-consensus computation of state updates.

2. Active replication: The ledger state after running a transaction is persisted only after the total ordering is established in the validation phase done by every node. This can be thought of as post-consensus validation of execution results.

In Fabric every transaction can be executed by a subset of peers, this is governed by the application specific endorsement policy. At the endorser after simulating the transactions a writeset(modified keys along with new values) and a read set(all read keys during execution) are created. The readset and writeset are then cryptographically signed by the endorser and sent back to the client as a proposal response. There is a possibility of endorsement policy not being satisfied when there is a high contention of operations accessing the same key.

2.2.3 Chaincode

Smart Contracts in Fabric are called **chaincodes**. It is the program code which implements the application logic. It can be written in general purpose programming languages like Go, Java or Node.js.

Chaincodes can directly access the state of the blockchain and can invoke other chaincodes assuming permissions and scope are correct. Chaincodes are executed in an environment which is very loosely coupled to the actual peer. This allows for having support for different programming languages for developing chaincode. The chaincode communicates via gRPC messages. The loose coupling is achieved by having the chaincode run in its own docker container.

Each chaincode can have multiple endpoints similar to REST endpoints on web servers. Chaincode endpoints can take any number of arguments and return a string or JSON output.

2.2.4 System Chaincode

They are special chaincodes which exist for the purpose of managing the blockchain system and internal parameters. Contrary to application chaincodes (chaincodes used to build applications), System chaincodes run on the same peer process and have access to the peer functionality directly. Some pre-installed chaincodes are the VSCC(Validation System Chaincode) and QSCC(Query System Chaincode).

2.2.5 Endorsement Policy

Endorsement policy is the set of rules that Fabric follows for the validation phase. It basically states the number of endorsements from peers to commit a transaction. Only administrators of the blockchain can set the endorsement policy, the chaincode developers will not be able to change it.

2.2.6 Ledger

The ledger component at each peer holds the ledger state on persistent store, it stores the transaction blocks as a set of append only files. The blocks arrive in a definite order hence the append only structure gives maximum performance.

2.2.7 Ordering Service

A subset of nodes in a Fabric network provide a total ordering on the transactions. They form the ordering service, a service independent of the peers which participate in execution. This decoupling of ordering service nodes allows consensus to be pluggable and modular. When a client has enough endorsements on a proposal it assembles a transaction and submits it to the ordering service. The ordering service then establishes a total order on all submitted transactions by a form of atomic broadcast of all the endorsements. Multiple transactions are grouped as a block and a hash sequence of blocks is created. A crucial feature of Fabric is that the ordering service does not execute any transaction on its own and does not maintain the state of the system. This allows for the consensus to be modular.

2.2.8 Channels

Another interesting feature of Fabric is the support for essentially multiple blockchains within the same context, which means they are connected to the same ordering service. Every such blockchain is called a channel. They allow blockchains to be shared between a subset of the parties. In a supply chain scenario, parties interacting with each other for orders directly might wish to have a channel between them.

2.3 SECRET SHARING

Secret sharing is a technique for distributing a secret amongst a group of participants, each of whom is allocated a share of the secret. The secret can be reconstructed only when a sufficient number of shares of different types are combined together. Individual shares do not reveal any information of the secret on their own. In general, given an t-share, at least $t + 1$ parties are required to reconstruct the corresponding secret.

Shamir secret sharing[31] is a popular secret-sharing scheme where t out of n secrets are required to recover a secret. This is based on an idea that a unique $t - 1$ degree polynomial can be fit to any set of t points that lie on the polynomial.

2.3.1 AVSS

In a verifiable secret sharing (VSS) protocol the dealer shares a secret with a set of n parties such that t+1 honest parties can reconstruct the secret. It is an essential component of secure multiparty computation (MPC) protocols, used both for generating random preprocessing

elements and for accepting secret-shared inputs from untrusted clients. Verifiability implies that if any party receives its share, then every correct party also receives a valid share. This is an important property for MPC applications since every honest party requires the share to be available to them. Polynomial commitments by the dealer allow the parties to validate the shares sent by the dealer in an independent manner. Asynchronous protocols have to deal with crashed nodes and slow nodes. Since there is no way to distinguish between these situations additional redundancy is required while distributing the shares so that parties with invalid shares can recover their shares by interacting with other parties. The protocol needs to satisfy correctness, confidentiality and agreement.

- Correctness: If the dealer is correct then all parties P_i will output share $\phi(i)$ where ϕ is the random polynomial and $\phi(0)$ is the secret.
- Confidentiality: If the dealer is uncorrupted, then adversary will learn nothing about the secret except for the output of the shares of the corrupted adversaries.
- Agreement: If any correct party receives output then there is a unique degree-t polynomial $\phi\prime$ such that each correct party eventually outputs $\phi\prime(i)$.

2.3.2 HoneyBadger of AVSS Protocols

Honeybadger of AVSS Protocols (HbAVSS)[32] provides linear amortized communication overhead even in the worst case of tolerating $f < n/3$ Byzantine faults. Compared to the previous work in the area which would provide optimal performance at only $f < n/4$ or a fallback to quadratic overhead in case of Byzantine faults.

The main idea behind HbAVSS is borrowed from HoneybadgerBFT[33] where a technique called encrypt-then-disperse was used. The secret shares are encrypted before they are dispersed. This guarantees robustness and efficiency as secrecy need not be provided.

In an AVSS protocol the dealer receives an input secret which needs to be shared among n parties. The n parties output a share for some t-degree polynomial. The shares from AVSS are used as input for every party in MPC computations.

2.4 MULTIPARTY COMPUTATION

The setting for multi-party computation involves n parties $P_1, P_2....P_n$ who compute a function f which takes k secrets$(S_1, S_2...S_k)$ as input parameters. The goal is to compute $y = f(S_1, S_2....S_k)$ while making sure correctness and privacy are preserved.

1. Correctness : The correct value of y is calculated, similar to what the value would be if it were to be computed by a single party with all the inputs.

2. Confidentiality : There should be no other information be released except the output y of the MPC computation.

2.4.1 Preprocessed Elements

Preprocessed elements simplify some of the computations involved in MPC. Preprocessed elements are usually generated prior to the actual MPC computation.

- Beaver triples[34] : Beaver triples are used for evaluation of multiplications in an MPC setting with t-shares. Each beaver triple is a triplet of t-shares $[x]_t$, $[y]_t$ and $[z]_t$ such that $z = x * y$. We can use these triples to evaluate multiplication of shares $[a]_t$ and $[b]_t$ by making use of the following equations.

$$\begin{aligned} M &= Reconstruct([a]_t - [x]_t) \\ N &= Reconstruct([b]_t - [y]_t) \\ [ab]_t &= M * N + M * [y]_t + N * [x]_t + [z]_t \end{aligned} \tag{2.1}$$

- Random bits : For many MPC applications, we need access to shares of a random bit. That is, given a polynomial $P(.)$ such that $P(0) \in 0, 1$, each party i will have the share equal to $P(i)$. Random bit shares can be generated directly using random value shares. Given shares $[r]$ of a random value $r \in Z_p$ we can generate a random bit through the operation $\frac{1}{2}(\frac{[r]}{\sqrt{Reconstruct([r]*[r])}} + 1)$.

CHAPTER 3: RELATED WORK

There have been prior attempts to have private data on blockchains as of this writing. Some of the solutions rely on having encrypted data on the blockchain and having the key managed by the party which owns the data.

3.1 HYPERLEDGER FABRIC CHANNELS

A Direct and simple solution as it a standard feature of Hyperledger Fabric. This allows parties to share a private ledger among themselves and prevent access from other parties in the network. However all the parties in the channel have full access to the ledger. For an application like a sealed-bid auction the bids of all the parties would be visible in the system and hence prevents to provide a finer level of privacy where nobody except the party can access the secret.

3.2 ZERO KNOWLEDGE PROOFS

Zero Knowledge proofs allow provers to establish a statement with other parties without revealing any additional information. The party which has the secret would run a smart contract on its own and prove to the others that the contract ran successfully and correctly. This works in a setting where only one of the parties holds secret data.

3.3 HAWK PRIVACY PRESERVING BLOCKCHAIN

Hawk is a decentralized smart contract system that retains privacy of the transactions from the view of the public. It extends the smart contract programming model and a programmer need not know or use additional cryptographic tools while writing a private smart contract. The programming model adds constructs to define and differentiate between public and private data. They present a concept of a manager who would know the users inputs but would not disclose them. The manager need not be trusted and could be caught and reprimanded(loss of deposit) in case of deviation from protocol. They open up discussions about future work where **MPC could be used to build the manager**. Cloaking Fabric would be along the lines of the manager described in Hawk.

3.4 SOLIDUS

Solidus[35] is a system to have confidential transactions on public blockchains. They leverage Zero Knowledge Proofs along with the concept of Oblivious-RAM. Oblivious-RAM hides the memory access patterns of a program without changing the input and output or the algorithm itself. Solidus can hide the details of the transaction and the participants by using publicly verifiable Oblivious-RAM which combines Zero Knowledge Proofs with Oblivious-RAM to hide identities of the individual participant. Solidus is designed for settings where every transactions would depend on secrets of one participant.

3.5 ENIGMA

Enigma[36] is a system that supports computations on private data in a decentralized setting. Enigma is more of an off-chain privacy framework as it runs a modified version of secure MPC among parties different from the parties of the blockchain. It is meant to be used to offload privacy protected computations by smart contracts on existing blockchains like Ethereum. Enigma uses a variant of the popular SPDZ[37] protocol for the Multi Party Computation, SPDZ is not a robust protocol and would halt on failures.

3.6 CALYPSO

Calypso[19] is a decentralized private data management system for blockchains. The secrets are encrypted and stored as a whole on the blockchain. However the access control and identity management happens via a different network. The symmetric key required to decrypt the on-chain secret is secret shared among the participants of this side network. The access control primitives they provide are fairly substantial and can cover most uses cases including transfer of ownership of data. However since they are not storing the actual shares of the secret but storing the encrypted data. The applications are restricted to a store and retrieve kind, MPC operations are not possible.

3.7 MPC JOINS THE DARK SIDE

Cartlidge et al. present[38] a solution for dark pools in financial markets. Dark pools in financial markets which are pools of potential buyers and sellers of a service that want to purchase or sell in large volume without indicating their interest so as to not inflate the market value. Dark pools traditionally employ third party trusted brokers to handle

the sales and orders. In case of a failure mode of a trusted third party like leaking the orders beforehand, the market rates tend to fluctuate and cause catastrophic losses for the clients. They implement a solution for this using Scale-Mamba MPC toolkit. They present a Continuous double auction where they arrange bids in descending order and offers in the ascending order to form a Limit Order Book such that the highest bid and lowest order form a spread and decide what incoming bids and orders would be matched to transact. Scale-Mamba implements an actively secure MPC protocol which guarantees security up to one failure, which means on detecting a failure the MPC computation stops.

3.8 STRAIN

Strain[39] by Blass et al. is an auction protocol for blockchains which guarantees bid. Strain use Zero-Knowledge proofs to do pairwise comparison between bids and publishing them on the blockchain. Strain does not utilize secret sharing or MPC primitives hence caters to a weaker adversarial model than MPC where the adversary is aware of the order of bid by amount. This helps them achieve an optimal complexity of $O(n)$ and $3blocks$, which makes it perfect for permissionless blockchains. They also mention in the paper that Hyperledger Fabric would potentially be a good platform for secure auctions however did not have anonymity guarantees provided.

3.9 SUPPORTING PRIVATE DATA ON HYPERLEDGER FABRIC WITH SECURE MPC

The paper[40] By IBM Research presents a system to store secrets on a blockchain and perform MPC on the secret data. They use Hyperledger Fabric as the blockchain and write a chaincode to achieve the same. Parties store the data on the ledger encrypted with their own secret key, when it needs to use the data for a transaction the party would decrypt the ledger data and use it as an input to a transaction.

They describe the use of a "helper" server to set up communication channels for inter peer communication. They acknowledge the insecure nature of this and state that this is was done to demonstrate the possibility of such a system. The system stores encrypted data on the blockchain, the key to encrypt this data is held by special parties known as "privileged clients".

They discuss some properties that their system does not possess but a production system in the future might possess.

1. **Endorsement Policy** : They point out that it is important to align the trust model of the secure-MPC protocol with the endorsement policy i.e A protocol that can tolerate t adversarial parties should have an endorsement policy that requires at least t+1 peers to endorse a transaction. They also note that non-standard endorsement policies can be supported by using System chaincodes.

2. **Enforcing rogue peers** : A rogue peer is a peer that exploits the "execute and then endorse nature" of fabric to block transactions that do not align with the interests of the peer after executing the MPC protocol.

3. **Support multiple parties** : Their demo supports only three parties due to the underlying MPC protocol used which does not scale to more nodes.

This paper presents a lot of ideas that we will directly use in our implementation.

3.10 MATRIX

The work by Assi Barak et al. titled "An End-to-End System for Large Scale P2P MPC-as-a-Service and Low-Bandwidth MPC for Weak Participants"[41] discusses the importance of coordination in MPC i.e. getting multiple parties to start the MPC operation at the same time. They present MPSaaS(MPC as a service) which uses a system called MATRIX that helps coordinate the participation and running of MPC. MATRIX allows administrators to specify the name, description and the start time of the MPC operation along with the type of circuit. MATRIX is a centralized service and hence is a single point of failure for availability.

CHAPTER 4: DESIGN

4.1 PRELIMINARIES

4.1.1 Flexible Trust Model of Fabric

A property of Fabric is flexibility in its trust model which is the reason it very well may be deployed in various scenarios. The components of Fabric like the ordering service are pluggable and can be changed based on the deployment requirements. There are various decisions to be made while deciding the ordering service. Fabric presents a centralized orderer, a cluster-based orderer running a Crash Fault Tolerant consensus protocol like Raft[42] or Kafka[43] and even BFT protocols like BFT-SMaRt[26] can be used. Endorsement policy is the number of endorsements that are required for committing a transaction. This can be set on a per application level and allows for different security settings for each application e.g. `ANY('Org1.nodes')` would enforce that the transaction be signed by any one node from Organization 1.

4.1.2 Network Model

In the asynchronous network model there are no timing assumptions i.e. there is no bound on the time for messages to be delivered within. HoneyBadgerMPC and HoneyBadgerAVSS work in asynchronous settings as there are no timeout assumptions in the implementations of both. Endorsement policies of Fabric let us extend this to the endorser nodes as well where the first $2n/3 + 1$ valid endorsements would be sufficient to commit a transaction. At the time of writing, Fabric version 1.4 ships with a partially synchronous ordering service implemented in Raft[42] and Kafka[43]. However, the pluggable consensus model of Fabric allows future work to use HoneyBadgerBFT[33] which is BFT consensus in an asynchronous setting. A partially synchronous network model could be supported by using PBFT or Raft as the consensus module of the orderer. In this model the consensus algorithm would assume a δ bound for the delivery time of the messages. This δ could be something that is adaptive and changes as the protocol proceeds or could be static and set by the protocol developer.

4.1.3 Confidentiality

Confidentiality in our context implies that a secret shared by a user is not made public unless the user leaks it or it is used as a part of a chaincode application which reconstructs

secrets at some point. The output of an MPC operation is also private to the participants unless explicitly specified.

HoneyBadgerAVSS guarantees that if the dealer(client/party) is correct the adversary would learn nothing about the secret. HoneyBadgerMPC guarantees that apart from the output of the computation, the parties will not learn the other party's inputs. Endorsement policies on Fabric and (n, t) configuration on HoneyBadgerMPC enforce that an adversary cannot arbitrarily start reconstruction on a secret.

4.1.4 Integrity

Integrity in our context implies that the results of the arbitrary computations performed be true to the MPC program i.e. the result of a MPC computation running on Cloaking Fabric should be the same as the MPC computation running on HoneybadgerMPC. Integrity could also mean that the value of a secret or the metadata is not wrongly changed by an adversary. This is prevented by having write-once secrets which allow write only from the authenticated client. It is common practice to open-source chaincode and the MPC programs so that they can be verified publicly.

4.1.5 Availability

Availability is defined as access to the chaincode and the data without censorship. The ledger is replicated in Fabric and information such as metadata is available on every single node. HoneybadgerAVSS can tolerate t failures, hence even in the case of $f <= t$ failures, secrets can be recovered. Similarly, HoneyBadgerMPC is a robust MPC framework that tolerates t failures as well. Which means that the protocol does not halt even when there are $f <= t$ failures.

4.2 SYSTEM OVERVIEW

In this section we will discuss the overall design of Cloaking Fabric. Figure 4.1 shows the end-to-end working of Cloaking Fabric with respect to a single client who interacts with the system. The client interacts with Cloaking Fabric by sharing secrets which could be used as input to MPC, the MPC completes and publishes the result of the computation on the blockchain. The user has the ability to recover his secret input from the blockchain using private reconstruction.

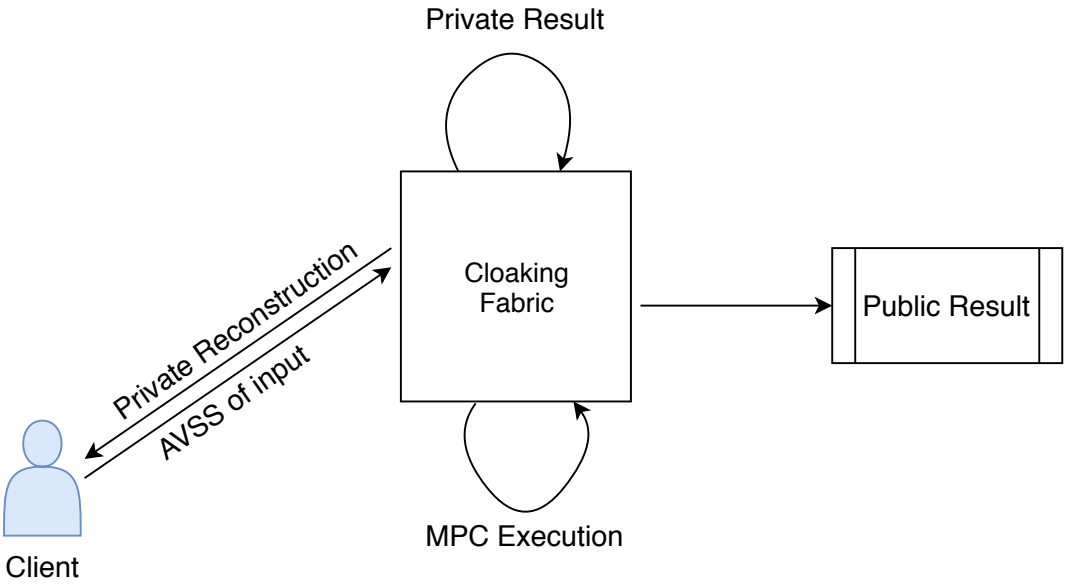

Figure 4.1: Cloaking Fabric in action

Cloaking Fabric is an extension for the existing Hyperledger Fabric system. We extend on it by building a system chaincode which interfaces with HoneybadgerMPC. Cloaking Fabric is roughly made up of the following components.

1. HoneyBadgerMPC : A general purpose toolkit for performing robust, scalable Multi Party Computation.

2. HoneyBadger System Chaincode : The system chaincode that interfaces with HoneyBadgerMPC and provides an abstraction for secret cells.

3. HoneyBadger Chaincode API : The API for chaincode developers to use to build applications that interact with secret cells.

4.3 HONEYBADGERMPC

HoneybadgerMPC is a multi-purpose MPC toolkit for running MPC applications as well as sharing and reconstructing secret data. HoneybadgerMPC is written in Python and we have a modified version of HoneybadgerMPC to interface with Fabric. It is important to use a robust MPC toolkit as Fabric works in a setting where nodes can arbitrarily crash and execution of chaincode should still go on seamlessly. If we use a non-robust MPC toolkit the MPC execution would halt on a single failure, any partial subset endorsement policy would turn out to be redundant.

These are some of the components of HoneyBadgerMPC that we build on.

- **HbAVSS** : An AVSS protocol that robustly tolerates up to t adversaries and allows for constant size commitments. AVSS uses sockets to communicate with other peers and hence we had to add modify the peer container to support the communication between AVSS peers.
- **Robust Reconstruction** : HoneyBadgerMPC allows for robust batch-based reconstruction of secrets. The interface of reconstruction is natural and easy to use and uses some of the same network primitives as HbAVSS.
- **Preprocessing** : To make certain operations faster, HoneyBadgerMPC uses precomputed random bits and/or beaver triples during execution. HoneyBadgerMPC has a comprehensive offline phase that generates and replenishes these preprocessed elements.
- **MPC context** : MPC context allows multiple parties to compute a function together by running the same code but have access to different shares of the data. This is implemented using asyncio and every operation that requires network communication is awaited on.

4.4 ARCHITECTURE

The Fabric ledger is shared by all the participants in the network and therefore they all share the same ledger state. For Cloaking Fabricwe need state which is private to a particular peer. Every share of a secret should be held by a unique peer to maintain the reconstructability property of the shares. If the shares were to be stored on the same ledger, all shares would be accessible to an adversary who could reconstruct the secret at any point. The shares would be stored in the individual nodes private state which for us is a Level-DB instance. Figure 4.3 dissects the node and lists the components layer by layer.

The metadata i.e. the public properties of the cell, should be stored on the public ledger and should be consistent among nodes. Figure 4.4 shows the network level view of the Cloaking Fabric. The public ledger is always in sync and agreed upon by the nodes but the private state is maintained independently by every node.

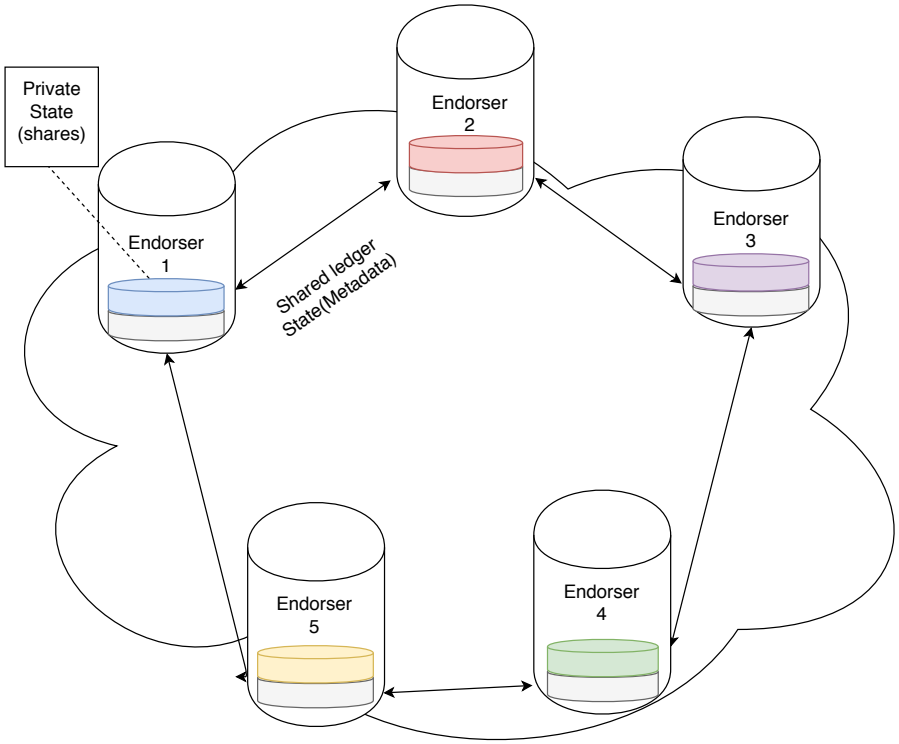

Figure 4.2: System Architecture - Network of Endorsers

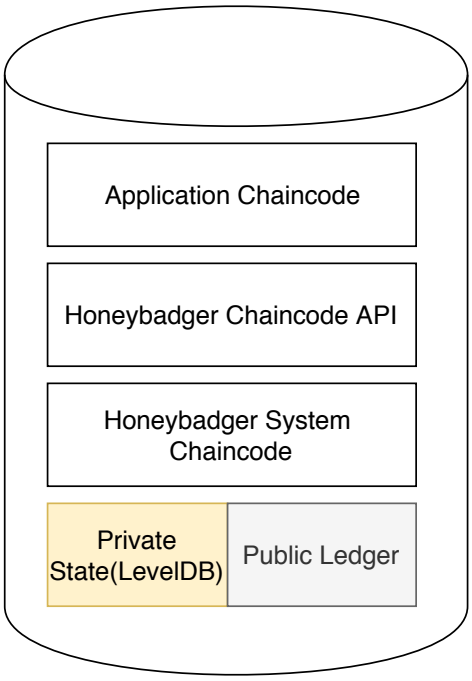

Figure 4.3: System Architecture - Components

4.5 HONEYBADGER SYSTEM CHAINCODE (HONEYBADGERSCC)

HoneyBadgerSCC is Fabric system chaincode that essentially interfaces with HoneybadgerMPC to provide an abstraction of secret cell. A secret cell is a **write once** structure that can hold a single value of secret data and its metadata is stored on the public ledger. A secret cell has the following properties.

- `CellName` : The name of the secret cell. It is the identifier for the secret cell and this value is used for all input and output fields which use secret cells. The cell name is set by the application and it is shared with the client who uses the cellname as a parameter while running AVSS on his secret input. `CellName` is also the primary key of the secret cell and cannot be changed in the lifecycle of the secret cell.

- `IsWritten` : A boolean value which indicates whether a cell has been written by the intended writer yet. The flag acts as a barrier for moving into the MPC phase of the application chaincode. This is updated by the system chaincode when the respective share for a given secret cell is received by the node and is validated to be correct.

- `WriterKey` : The writer key is the main access control primitive for Cloaking Fabric. The client interacting with the application passes their public key as an argument to the application chaincode endpoint. The application chaincode then creates and sets the `WriterKey` to that of the communicating client. When the client has to AVSS the values they provide valid certification to the system chaincode. The `WriterKey` when set to null indicates that no client can write to a secret cell. This is especially useful when the output of an MPC computation is a secret cell.

- `IsOpen` : A boolean which indicates if the secret data has been reconstructed yet or not. The `IsOpen` field is initially set to false but it is set to true once the reconstruction completes and the node receives the opened value.

- `Value` : This holds the value of the cell once the secret data has been reconstructed. Once the value is made public it is irreversible as the value would be known to all the nodes. The value field is initially empty and is only set after the reconstruction is completed by opening a share.

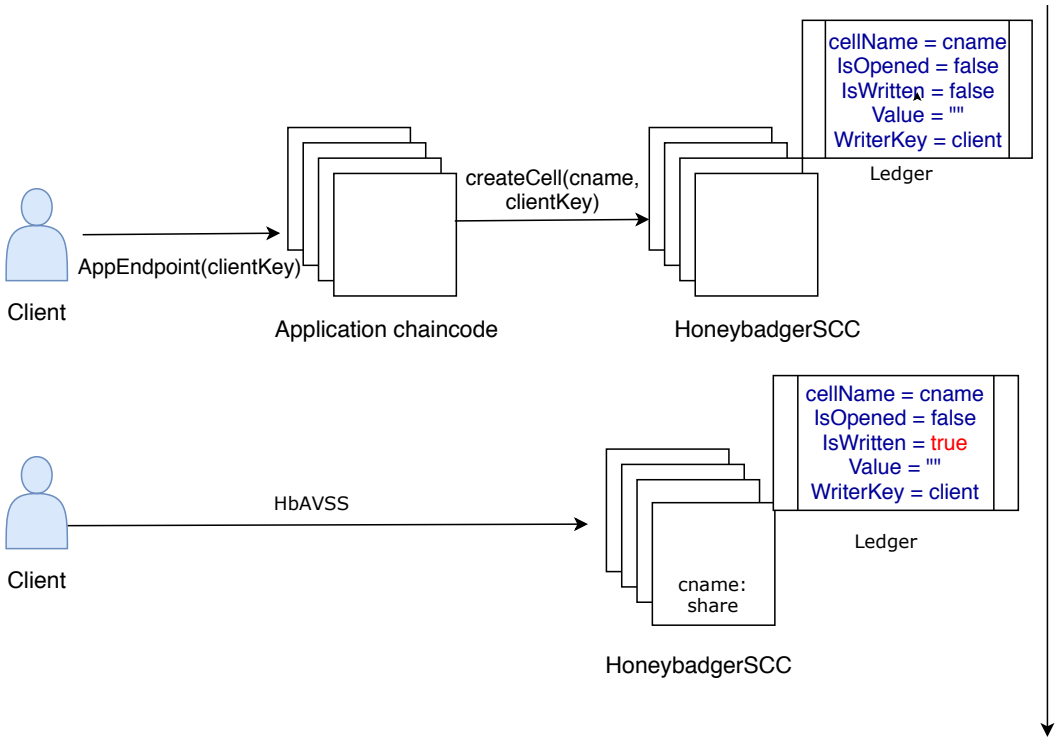

Figure 4.4: An illustration of a workflow involving secret sharing

4.6 OPERATIONS ON SECRET CELLS

4.6.1 Share Secret

Figure 4.4 illustrates the typical workflow during secret sharing. The client interacts with a chaincode endpoint and may be asked for input to a secret cell in some scenarios. The Application chaincode provides the client with the name of the secret cell and creates an entry for the secret cell by sending a request to HoneybadgerSCC.

The writer of a cell (application sets the public key of the writer) creates shares of the secret input and starts HbAVSS as a dealer to the other endorsing nodes. The endorsing nodes on completing HbAVSS successfully would store the share to their private state and update the cell's `IsWritten` to true. Depending on how many endorsers set the state and endorsement policy of the system, the transaction will be committed and the cell can be used in applications. In any event that the AVSS does not complete either due to a malicious client(dealer) or more than supported node failures($f > t$), the secret is not shared and `IsWritten` remains false.

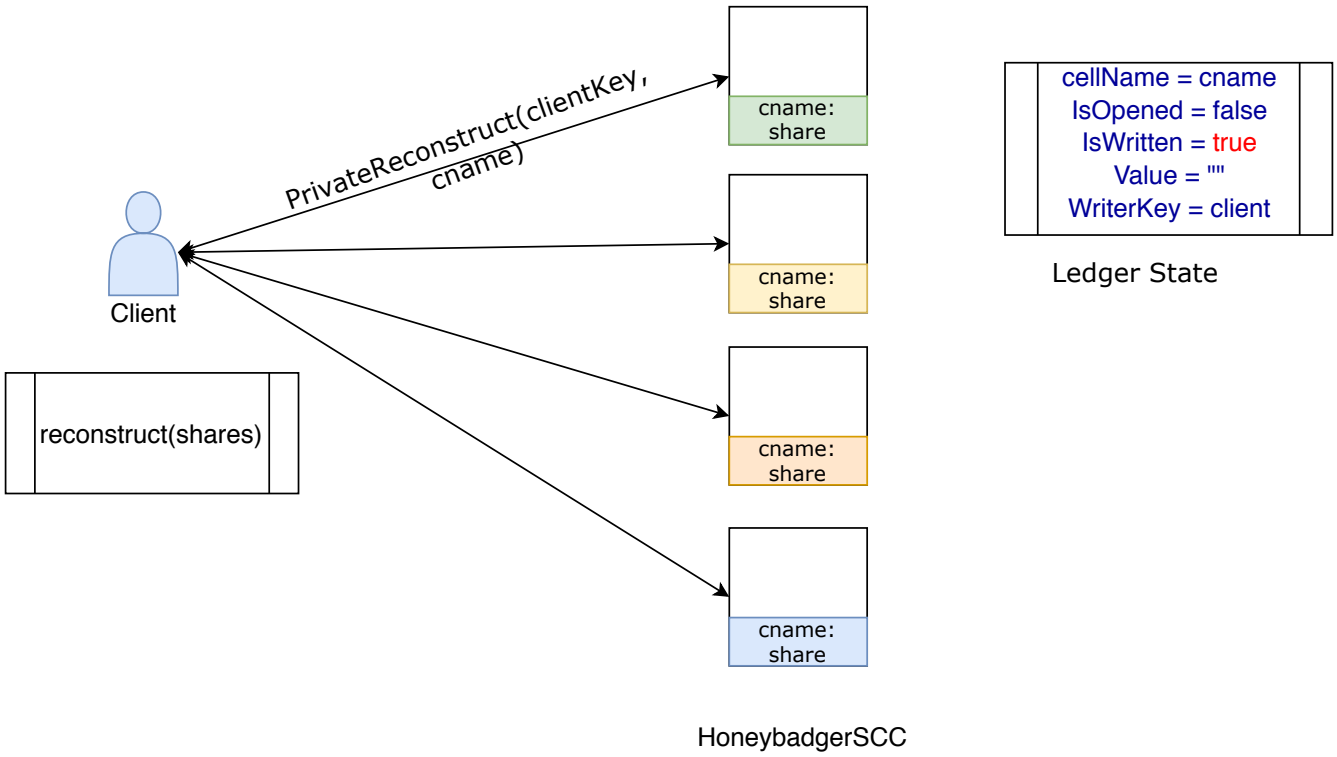

Figure 4.5: Private reconstruction of a cell

4.6.2 Private Reconstruction

Private Reconstruction is a primitive provided by Cloaking Fabric for scenarios when the client wants to retrieve a previously shared secret from the system. The client would sign a message with his private key and request for the individual shares from the endorsers. The endorsers after validating the identity of the user would return the shares for the particular cell. The client then runs reconstructions provided by HoneybadgerMPC locally and reconstructs the secret. Figure 4.5 shows the working of private reconstruction.

4.6.3 Public Reconstruction

In public reconstruction all endorsers participate together to reconstruct the secret from the shares they individually hold. Public reconstruction is used as a primitive in MPC operations and is useful for applications where an input has to be private for a certain duration of time and then needs to be revealed after satisfying a condition. Application chaincode would decide when it is time to reconstruct a secret, this would usually be after a certain number of inputs are available to be opened or a certain duration of time has elapsed since the creation of the instance. Figure 4.6 illustrates the working of Public Reconstruction.

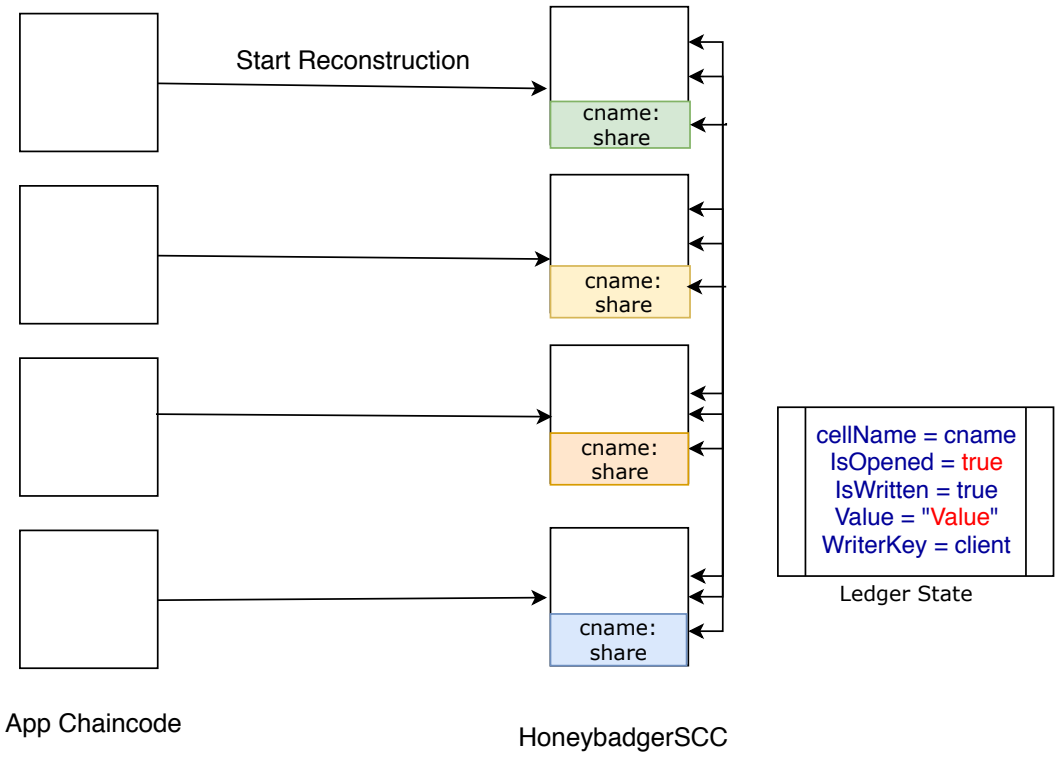

Figure 4.6: Public reconstruction of a cell

Public Reconstruction works in two phases from the point of view of application chaincode.

1. Application chaincode calls `reconstructSecret` API call to start the reconstruction phase. This returns true if the reconstruction was started successfully.

2. Application chaincode then calls `GetCellMetadata` to retrieve the metadata of the cell. It then checks the `IsOpen` attribute to check if the reconstruction was successful. and proceeds to use the reconstructed `Value`

4.6.4 MPC Operation(mpcOp)

`mpcOp` is a function to start MPC on the nodes. `mpcOp` takes a list of secret cells as input, the type of the MPC operation as an argument, instance name of the MPC operation and starts the MPC operation on the respective nodes. MPC operation is called by application chaincode when a certain condition is met e.g number of inputs, time elapsed, number of blocks in the blockchain. MPC operation only starts the MPC on the nodes, it sets a flag and result once the operation is complete. `checkMPCResult` is able to retrieve the result from the MPC operation.

```
func ReadAorB(choice) {
    if readToken.valid() {
        readToken.use()
        if choice == "A" {
            return readSecret("A")
        } else {
            return readSecret("B")
        }
    } else {
        return Error("No read token")
    }
}
```

Figure 4.7: Demonstration of breach in confidentiality due to speculative reads

4.7 SUPPORT FOR BARRIERS

4.7.1 Speculative Execution in Fabric

In the previous section we described the architecture of Fabric and how it follows a execution paradigm different from other blockchains. Executing a transaction first and then deciding the order and validity of the transaction based on read-write conflicts causes the execution to be speculative in nature. Although this sort of architecture allows for a higher throughput in general due to execution to be done by a subset of nodes, down side is making sure side effects from not committing a transaction do not breach the confidentiality or integrity of the system. This behavior is documented in the Fabric paper [15] and detailed in work from IBM Research[44].

4.7.2 Side Effects from Speculation

A side effect occurs when a transaction performs an irreversible operation that cannot be undone in the validation phase in the event the transaction is not committed. A side effect could be something as simple as returning the result of an uncommitted transaction.

Consider the above example in figure 4.7. The function here allows the user to read a secret either A or B but not both. In the event that there are two simultaneous calls to the function in Fabric i.e before one of the transactions are committed, there is a chance that a malicious user would be allowed to read both the secrets. These hazards can be exploited in a variety of settings if the application does not handle it correctly.

- Free reads in Pay for access: a client may purchase a “token” to read one record

```
func sampleApplication() {
    A = reconstructSecret("a")
    B = reconstructSecret("b")
    if checkValidity(A) && checkValidity(B) {
        applicationLogic()
    } else {
        return Error()
    }
}
```

Figure 4.8: Demonstration of a hazard in application chaincode using Cloaking Fabric

from the database, e.g. to download an e-book. Speculative reads would let a client effectively double spend that token.

- Escape Audit Logs : Access logging[16] is a pragmatic approach to privacy by allowing clients easy access to a large data source but audit every time a client reads data, this enforces accountability on the reader of the data as they can be tracked if they read the data without authorization. This sort of speculative execution can be exploited by malicious clients to read data without being logged.

- Zero knowledge proof or related cryptography gadgets: Zero knowledge proofs and other similar class of protocols in cryptography there is an established pattern where the "proof verifier" is only allowed to choose only one value to read. In such a scenario speculative reads can be used to read more than required amount of data.

Naturally such issues could arise in applications using Cloaking Fabric. We have irreversible operations in reconstructing private data and MPC computations.

In Figure 4.8, The application chaincode attempts to reconstruct two related secret cells for a transaction and then checks the validity of the cells. In the event that the second cell does not exist the transaction is not valid, however the first secret would have irreversibly been reconstructed.

4.7.3 Barriers

To address the above problem of speculative execution hazards we have a couple methods to serve as barriers for application programmers. Barriers essentially force the application logic to wait for the prior condition to be updated and set on the blockchain. The barrier fields are only updated in the blockchain if the endorsement policy is met. The following fields can be used as barrier fields. A solution to the hazard in Figure 4.8 using barriers is show in Figure 4.7.3.

```
func sampleApplication() {
    A = getSecretCell("a")
    B = getSecretCell("b")
    if A.isOpen && B.isOpen {
        // running application logic when both the cells are open
        applicationLogic()
    }
    else if A.isWritten && B.isWritten && additionalAppCondition {
        // opening the cell when both the cells have been written
        A.reconstruct()
        B.reconstruct()
    } else {
        return Error()
    }
}
```

Figure 4.9: Demonstration of the use of barriers

1. `IsWritten` : this field is updated only when the secret is shared by the client successfully. That involves AVSS being run by $(2/3n)+1$ nodes, then the secret cell structure needs to be updated on the blockchain which requires $(2n/3)+1$ signatures from the endorsers.

2. `IsOpen` : similarly this is updated only when the share is opened successfully, requires at least $(2n/3)+1$ nodes to complete the opening.

3. `checkMPCResult()` : A function in the Honeybadger chaincode API which returns if the MPC result is available or not. Since the MPC operation is asynchronous, this is helpful in determining if the MPC operation has completed or not.

Application chaincode must be written in a way that they return no other information if the barrier condition is not satisfied and the client is expected to be polling to check the result of an operation.

CHAPTER 5: APPLICATIONS AND EVALUATION

In this section we present some applications built using Cloaking Fabric. We then evaluate the performance of these applications and compare Cloaking Fabric with related work.

5.1 APPLICATIONS

We present three different applications to show the working of Cloaking Fabric. First we present a (1) secure **Rock Paper Scissors** game which utilizes secret cells to reduce the number of phases compared to traditional implementations. We then demonstrate the capability to do MPC on secret cells with three applications (2) a **secure sealed bid auction** and (3) **secure linear regression**.

5.1.1 Rock Paper Scissors

We present a new take on the classic Rock Paper Scissors game to showcase an application that interacts with secret cells. The user inputs will be secret shared using HbAVSS and stored in secret cells. Only when both the parties have shared their input the reconstruction phase can start, after which the inputs are made public for the result to be computed and published.

Consider a traditional rock paper scissors smart contract[45] which does not have the ability to access or store private data. The users provide a commitment of their move(which could be the move hashed along with a random seed). Once both the players input their moves. One of the players starts the reveal phase by revealing his commitment(providing the move and the random seed individually to the chaincode). The second player has a limited time to reveal his move after the first player has revealed his move. Once both the users have revealed their move the result is computed and published. If the second player fails to reveal his move in the allocated time, he forfeits.

The above approach is divided into two distinct phases (1). Users input their move and (2). Users reveal their move. A failure mode here would be in phase (2) where a malicious player learns that his move is not the winning move after the other player's move is revealed and decides to exit the protocol without revealing their move. To counter this a timeout is added after the first player reveals their move, else a penalty is imposed on the player.

Timeouts have issues of their own. In blockchains where transactions rates are low, time-

```
1       if fn == "createGame" {
2           //creates a game
3            memcell := getMemoryCellName()
4            createCell(memcell, userKey)
5            // create a memory cell for the user move
6            game := createGame(gameName, userKey, "None", memcell, None)
7            game.save()
8            return Success(memcell)
9        } else if fn == "joinGame" {
10           memcell := getMemoryCellName()
11           createCell(memcell, userKey)
12           game := GetGame(gameName)
13           game.user2Key = userKey
14           game.user2Move = memcell
15           game.save()
16           return Success(memcell)
17       }
18       else if fn == "getActiveGames" {
19            // returns the active game lobby
20           return getActiveGames(stub)
21       } else if fn == "getCompletedGames" {
22            // returns the completed games scoreboard
23           return getCompletedGames(stub)
24       } else if fn == "startRecon" {
25            game := GetGame(gameName)
26            cell1 = GetCellMetadata(game.user1Move)
27            cell2 = GetCellMetadata(game.user2Move)
28            if cell1.IsWritten && cell2.IsWritten {
29                // reconstruct only when both values are available
30                reconstructSecret(cell1)
31                reconstructSecret(cell2)
32                return Success("Started Reconstruct");
33           } else {
34                return Failure("InputNotShared")
35           }
36       } else if fn == "getResult" {
37            // result of the game if the moves have been completed
38            cell1 = GetCellMetadata(game.user1Move)
39            cell2 = GetCellMetadata(game.user2Move)
40            if( if cell1.IsOpen && cell2.IsOpen ){
41                // computeResult applied RPS logic on the cells
42                return Success(computeResult(cell1, cell2))
43           } else {
44                return Failure("Input not Opened")
45           }
46
47       }
```

Figure 5.1: A simplified version of the Rock Paper Scissors Chaincode

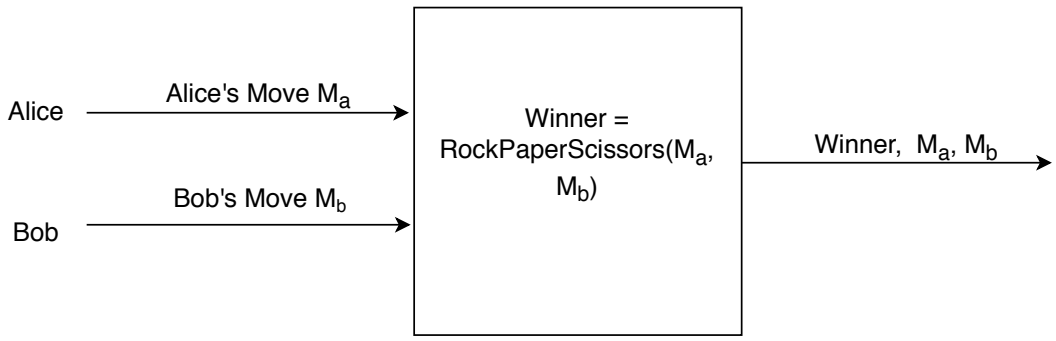

Figure 5.2: Ideal functionality of Secure Rock Paper Scissors

outs increase the total game time and are a hindrance to legitimate users who fail to reveal their move on time due to network delays. More importantly it makes the players interact multiple times which makes it less intuitive.

To understand how Rock, Paper Scissors would work with a trusted third party we can observe the ideal functionality in Figure 5.2. In our solution the chaincode would be the trusted third party which reconstructs the user inputs only after both the users have shared their move.

The user who creates the game would interact with `createGame` endpoint in Figure 5.1. `createGame` creates an instance of the game and a secret cell to which the user will share his move. The user is then expected to share his move via HbAVSS. Potential players view active games by invoking the `getActiveGames` endpoint, they then use the name of the game to `joinGame` and share their move. Any entity can ask for the result for the result to be computed by invoking `startRecon` on the game instance, this would start reconstruction on the cells only if both the cells are available. Once the reconstruction is completed the result would be available by invoking `checkResult`. In our approach the users can just "fire and forget" their input and need not wait for the other user to make their move to start the reveal phase. Result of previous games would be available by invoking `getCompletedGames` endpoint. This simple application demonstrates the power of a blockchain system with access to managing secret data and could easily be used for other applications such as bidding and betting. The unique programming model of Cloaking Fabric helps us build useful features like lobbies and scoreboard for the games.

Here, we make sure the fundamental properties are satisfied.

1. **Confidentiality** : The user's input should not be made public until both the users submit their moves. This is guaranteed by not starting the reconstruction until both

```
async def auction_mpc(context, bids):
    max_bid = bids[0]
    for i in range(1, len(bids)):
        t = await max_bid.gt(bids[i]) # t = max_bid < bids[i]
        max_bid = (await (t.mul(max_bid.sub(bids[i])))).add(bids[i])
        # max_bid = (t* (max_bid - bids[i])) + bids[i]
    return max_bid
```

Figure 5.3: MPC program for Secure Auction

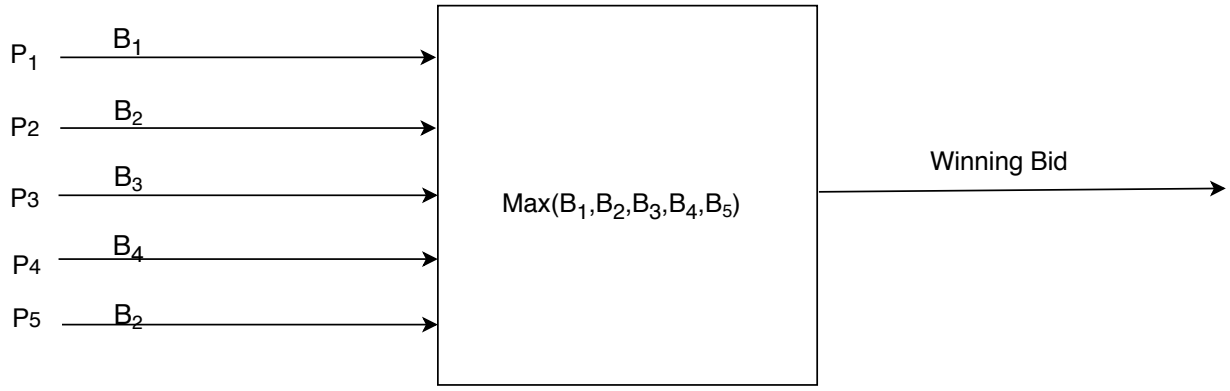

Figure 5.4: Ideal functionality of secure Auction

inputs are written.

2. **Integrity** : Integrity implies that a user's move should not be changed after reconstruction has started. Since the secret cells of Cloaking Fabric are write once, this property is guaranteed.

3. **Availability** : Availability should guarantee that no party should be able to halt the computation of the result after participating in the game. By avoiding a reveal phase we ensure that no malicious party cannot halt the execution of the contract.

5.1.2 Secure Auctions

Auctions have always been a cornerstone application for demonstrating MPC ever since the first live real use case of MPC was used for a beets auction in Denmark[46]. The ideal functionality of a secure auction using a trusted third party is shown in Figure 5.4. A corrupt trusted third party especially in auctions has too much power to change the natural execution of the auction, the third party could collude with a participant, they could also reveal incorrect information to drive the price, hence a decentralized auction is highly desirable when the stakes of the auction are high. It is often a requirement for auctions

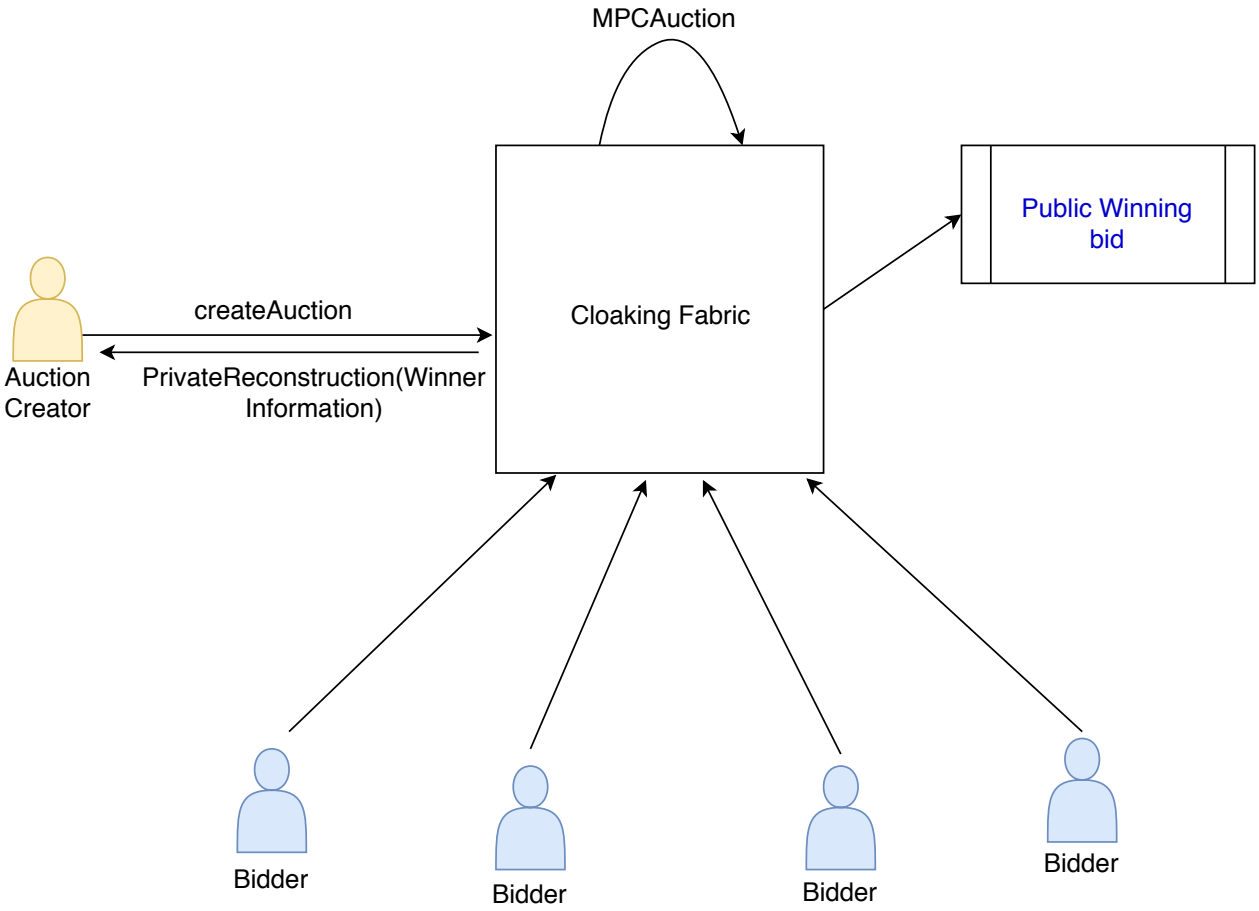

Figure 5.5: Workflow of a Secure Auction

where the bids of the participating members of the auction be private. This is done so as to avoid malicious behavior of the participants who could in theory collude with a subset of the parties, exit the auction abruptly and disturb the flow of the auction. Making an auction "decentralized" replaces the trusted third party with MPC.

We present a first price auction where we implement an auction as an MPC program and deploy it on Cloaking Fabric. The chaincode can run multiple instances of the auction, it identifies the auction by a name provided by the creator. The creator would then provide additional parameters such as participant list and an end condition such as maximum number of bids or a certain duration of time as the end point. Similar to the Rock, Paper, Scissors example the participants could view the auction via a lobby and decide to join the auction. Figure 5.5 presents the workflow of an auction with multiple clients and Figure 5.9 shows the chaincode snippet for the same. The creator of the auction specifies the total number of bids required to conduct the auction in `createAuction` and the auction MPC program starts once that condition is met. We implement the MPC program for the auction in Figure

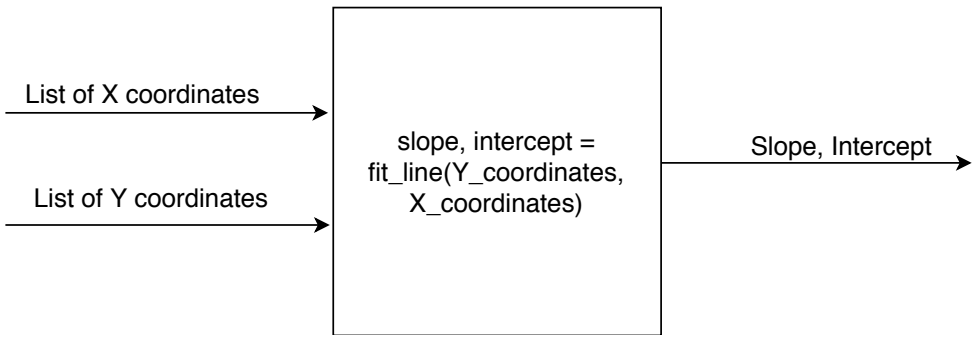

Figure 5.6: Ideal functionality of Linear Regression

5.3. Here we do sequential comparisons[47] to implement a *Max* function. In order to not reveal the order of bids we refrain from using "if/else" statements and open the max value only at the end.

The fundamental properties are satisfied here as follows.

1. **Confidentiality** : The bids of the users are required to be private and only the winning bid is made public, there is no information revealed about the relative order of the bids.

2. **Integrity** : Auction must compute the winner accurately, we rely on the underlying MPC program to work as advertised, since the code is open source it can be thoroughly inspected for correctness. We use FixedPoint integers of the range $[-2^{31}, 2^{31} - 2^{-32}]$ for the secret shared values. An adversary could enter a bid which is not in this range. To validate the bid we suggest future work to use Zero Knowledge Ranged proofs[48] to prove that the input lies in a particular range.

3. **Availability** : Parties should be able to participate in the auctions freely. The programming model allows for lobby style interfaces which could announce an auction well in advance so that all interested parties know about an auction in advance. Malicious users cannot halt the auction after making their bid.

5.1.3 Linear Regression

Linear Regression is a statistical technique of modelling the relationship between a dependent variable and one or more explanatory (independent) variable[49]. The relationships are modeled using linear predictor functions which who's unknown parameters are estimated from the data. Linear Regression is used in a lot of finance, health data and forecasting

analytics. The ability to perform Linear Regression with private data inputs would be invaluable to users of Hyperledger Fabric.

Given a dataset consisting of n data points with independent variables $\{x_1, x_2, \ldots, x_n\}$ and the corresponding dependent variables $\{y_1, y_2, \ldots, y_n\}$, a linear regression model can be used to fit a line between x and y.

The best-fit line can be parameterised with two parameters, m for the slope of the line and b for the y-intercept, such that $y = mx + b$. In general, a finite error ϵ_i will be present in our observations of the independent variables.

$$y_i = x_i + \epsilon_i \tag{5.1}$$

In order to find a best-fit model, we attempt to minimize the least-squares error of our model which is given by the following equation

$$J(m, b) = \frac{1}{2} \sum_{i=1}^{n} (y_i - (mx_i + b))^2 \tag{5.2}$$

A closed-form solution exists for the parameters m and b that minimize this error. However, the closed form solution involves matrix inversions which are not supported by HoneyBadgerMPC at this point. We instead resolve to using a gradient-descent based approach to find the best parameters.

In a gradient-based approach, we first randomly pick initial values for m and b as m_0 and b_0 respectively. We then iteratively do the following operations -

$$m_{t+1} = m_t - \alpha * \frac{\partial J}{\partial m} \tag{5.3}$$

$$b_{t+1} = b_t - \alpha * \frac{\partial J}{\partial b} \tag{5.4}$$

We use fixed-point arithmetic [50] for linear regression, and use beaver triples for multiplication. Random bits required for fixed-point arithmetic are generated and stored as part of a preprocessing phase.

The ideal functionality for Linear Regression is shown in Figure 5.6 and we implement it by replacing the trusted third party by an MPC program[51]. Figures 5.7 and 5.8 provide the chaincode for implementing Linear Regression. The chaincode interface is similar to that

of secure auction and differs in the number of parameters.

The fundamental properties are satisfied here as follows.

1. **Confidentiality** : To satisfy confidentiality the x and y coordinates of the point should not be revealed. The MPC program[51] does not open the x and y coordinates in their raw form. Only the slope and intercept will be opened.

2. **Integrity** : We use FixedPoint integers of the range $[-2^{31}, 2^{31} - 2^{-32}]$ for the secret shared values. An adversary could enter a point which is not in this range. To validate the bid we suggest future work to use Zero Knowledge Ranged proofs[48] to prove that the input lies in a particular range.

3. **Availability** : All parties should be able to participate in the Linear Regression freely and no client should be able to halt the execution. client in our model just "fire and forget" the input points and are not expected to participate further.

5.2 IMPLEMENTATION DETAILS

5.2.1 Go Plugins

We implement HoneybadgerSCC as a system chaincode and we opted to build it as a Go plugin which would allow it to be loaded dynamically in our test system. This helped us rapidly prototype and make changes and test them immediately. The standard technique of building system chaincodes involves bundling it with the other system chaincodes and rebuilding Fabric. We found this approach to be detrimental to prototyping and testing. For compiling the system chaincode we create a docker container of the same type as the final fabric peer and mount the file system and compile the image. We found this approach to reduce build times to a few seconds.

5.2.2 Test Network

We tested our system on the `first network` example network provided in the `fabric-samples`[52] repository. It uses a 4 peer system with a single certificate authority and a single node ordering service. In first network there are two organizations and each organization has 2 nodes. For our tests we have considered all nodes to be independent and distrustful of all other nodes.

```
type LinReg struct {
    ObjectType    string    `json:"docType"`
    Name          string    `json:"name"`
    EndPoints     int       `json:endPoints`
    Participants  []string `json:"participants"`
    X             []string `json:"X"`
    Y             []string `json:"Y"`
    M             int       `json:"M"`
    B             int       `json:"B"`
    Result        string    `json:"Result"`
}
```

Figure 5.7: Linear Regression Instance Structure

A comparison of properties of different systems			
Name	Support for private data	Availability	Arbitrary Computations
Cloaking Fabric	Yes	Yes	Yes
Calypso	Yes	Yes	No
HyperMPC(MATRIX)	Yes	No	Yes
Fabric	No	Yes	Yes
Scale-Mamba	Yes	No	Yes

Table 5.1: Feature Matrix of different systems

5.3 EVALUATION

In this section we evaluate the performance of Cloaking Fabric and identify bottlenecks and overheads. We also compare Cloaking Fabric with related work and discuss the different attributes and trade-offs.

5.3.1 Feature Comparison

Table 5.1 is a feature matrix of related systems. The properties discussed are

- **Support for private data**: A system supports private data if there is a mechanism

HbAVSS n=4, t=1			
k(number of values)	Time taken	Bandwidth at node	Bandwidth at dealer
1	4.72	3417	5244
100	15.81	297203	525839
1000	56.73	2987488	5276382

Table 5.2: Secret Sharing benchmarks

```
1         if fn == "createInstance" {
2             // Create an instance of LinearRegression
3             LinReg := &LinReg{linearReg, instanceName, endParams}
4             LinReg.save()
5             return Success("Created Instance"))
6         } else if fn == "getActiveLinRegs" {
7             return getActiveLinRegs(stub)
8         } else if fn == "getCompletedLinRegs" {
9             return getCompletedLinRegs(stub)
10        } else if fn == "addData" {
11            // returns memory cells id for the user input
12            LinRegInstance := getInstance(key, stub)
13            memcellX, memcellY := getMemCellNames()
14            createCell(stub, memcellX, args[1], "linReg")
15            createCell(stub, memcellY, args[1], "linReg")
16            LinRegInstance.X = append(LinRegInstance.X, memcellX)
17            LinRegInstance.Y = append(LinRegInstance.Y, memcellY)
18            LinRegInstance.save(stub)
19            return Success(memcellX, memcellY)
20        } else if fn == "runMPC"  {
21            if len(LinRegInstance.X) < LinRegInstance.EndPoints && len(
                  LinRegInstance.Y) < LinRegInstance.EndPoints {
22                return Error("Not enough points")
23            }
24            for i, _ := range LinRegInstance.X {
25                var cellX, cellY secretcellX
26                cellX = getCellMetaData(stub, LinRegInstance.X[i])
27                cellY = getCellMetaData(stub, LinRegInstance.Y[i])
28                if cellX.IsWritten == false || cellY.IsWritten == false{
29                    return Error("Unwritten cells")
30                }
31            }
32            cells := append(LinRegInstance.X, LinRegInstance.Y...)
33            mpcOp("linear_regression_mpc", LinRegInstance.Name, cells...)
34             return Success(byte("Started")
35        } else if fn == "getResult"  {
36            if checkMPCResult(stub, LinRegInstance.Name, "LinReg") == "None
                  " {
37                return Error("No Result yet")
38            }
39            LinRegInstance.Result = checkMPCResult(stub, LinRegInstance.
                  Name)
40            LinRegInstance.save()
41            return Success(LinRegInstance.Result)
42        }
```

Figure 5.8: Linear Regression Chaincode

```
1
2    else if fn == "runMPC"  {
3            Auction := GetAuctionInstance(auctionName)
4            for _, element := range Auction.Bids {
5                cell := getCellMetaData(stub, element)
6                if cell.IsWritten == false {
7                    Error("Unwritten Bids")
8                }
9            }
10           outputCell = createCell(getCellName(), Auction.creatorKey)
11           cells := (Auction.Bids)
12           mpcOp("auction", Auction.Name, outputCell, cells...)
13           return Success("Started MPC")
14       } else if fn == "getResult" {
15           Auction := GetAuctionInstance(auctionName)
16           if checkMPCResult(Auction.Name) == "None" {
17               return shim.Success([]byte("None"))
18           }
19           Auction.Result = checkMPCResult(Auction.Name)
20           Auction.save()
21           return Success(Auction.Result)
22       }
```

Figure 5.9: Auction chaincode

Cost of running the application			
Application Name	Number of bits	Number of triples	Rounds of communication
Rock Paper Scissors	0	0	2
Auction(N bids)	124N	187N	200N
Linear Regression	64 per epoch	19 per epoch	90 per epoch

Table 5.3: Application cost matrix

Running time for the applications n=4, t=1			
Application Name	Time taken(Cloaking Fabric) in seconds	Time taken(HBMPC) in seconds	overhead(in seconds)
Rock Paper Scissors	6.72	3.67	3.05
Auction(7 bids)	19.17	17.37	1.8
Linear Regression(100 epochs)	52.43	47.24	5.19
Linear Regression(1000 epochs)	469.85	477.8	7.95

Table 5.4: Running time of different algorithms

Running time for the applications n=4, t=1 with one failure			
Application Name	Time taken with 1 failures with Cloaking Fabric	Time taken with 1 failure in HBMPC	Overhead(in seconds)
Rock Paper Scissors	6.41	3.45	2.96
Auction(7 bids)	17.47	15.3	2.17
Linear Regression(100 epochs)	40.91	36.29	4.62
Linear Regression(1000 epochs)	393.24	387.30	5.93

Table 5.5: Running time of different algorithms with 1 failure

to store the secret of one party on multiple nodes but only the storing party would know and have access to the actual secret. All the systems in the list support secret data except the plain Hyperledger Fabric. Fabric channels do not qualify for this as the data is visible to all parties in the channel of multiple parties and single party channels would not be fault tolerant.

- **Availability** : Availability attributes to high fault tolerance, lack of single point of failure and high resilience. Availability is crucial for running applications on wide area networks with a large number of nodes. Scale Mamba implements a non-robust MPC protocol which halts on detecting failures, MATRIX provides a centralized service for coordinating MPC. This would be a single point of failure for the coordination service. On the contrary Cloaking Fabric is backed by a blockchain which would be replicated and provide high availability.

- **Arbitrary Computations** : Running arbitrary functions on the secret data would be the third metric to compare with. Calypso stores encrypted data on the blockchain and the key to access it would be stored as secret shares. Cloaking Fabric stores the secret data as secret shares which can be used as inputs to MPC programs. Any computation with respect to Calypso would involve reconstructing the secret, performing a computation and encrypting the data again. This would involve a trusted third party and would not be feasible.

5.3.2 Comparison With Prior Work in Fabric

Prior work by IBM Research[20] present a modification to Fabric to support private data. In the paper, they mention that their work aims to be initial proof of concept in this direction.

```
1    OutOf (3, 'Org1MSP.member', 'Org1MSP.member', 'Org2MSP.member', '
        Org2MSP.member')
```

Figure 5.10: Endorsement Policy Snippet

They mention the following shortcomings of their approach in the paper.

- **Endorsement Policy** : At the time of their writing Fabric did not provide enough freedom to specify an Endorsement policy to align with the trust model of the MPC protocol. Since then, Fabric has evolved to support a language to specify complex endorsement policies and Cloaking Fabric makes full use of it. For example, the endorsement policy in Figure 5.10 can be used in a $n = 4, t = 1$ setting for the `first network` sample to require endorsements from 3 peers.

- **Point to point communication** : They opt for a centralized server to establish communication channels between nodes for the point to point communication between MPC peers. In Cloaking Fabric we use system chaincode which have the privilege to set up sockets and mange their lifetime.

- **Rogue Peers** : In their implementation a rogue peer could deny to endorse an MPC operation after learning the value of the computation. Cloaking Fabric uses endorsement policies which are aligned to the MPC framework and can tolerate rogue peers which do not endorse the result after executing it. On a side not the result of the MPC operation could also be made private.

To summarize we address all the issues they describe in their implementation and provide a comprehensive programming model with the abstraction of a secret cell.

5.3.3 Comparison of Cloaking Fabric's Programming Model with MATRIX

MATRIX provides a centralized architecture for coordinating MPC protocols. MATRIX comes bundled with an administrator component which is used to create, manage and start MPC protocols. A web based user interface is provided to the administrator where they can specify the title, description, registration time and type of result(MPC operation) and create an entry which is then visible to all potential participants. It is apparent that this is suitable for large scale deployments of simple MPC computations such as number surveys.

Compared to MATRIX, Cloaking Fabric is decentralized and runs as a part of a blockchain. The chaincode based programming model allows developers to build customized applications.

As we saw in Section 5.1 a variety of non-trivial applications are possible and follow the same pattern as chaincode applications, thus giving developers freedom to implement more comprehensive applications.

5.3.4 Experiment Setup

We run our experiments on a single server having 32 cores of `Intel(R) Xeon(R) CPU E5-2620 v4` at 2.10GHz with 120Gb of RAM. We run every single component(node, orderer, certificate authority) on separate docker containers and ensure they run on different physical cores. We run each of the above described applications in isolation and we assume pre-processed data(bits, triples) are readily available. For every experiment we take three runs and average the measurements over the three runs. For all Cloaking Fabric measurements, we measure the total time from when the client makes the request to the chaincode to when the result is available at the client.

5.3.5 Performance of Cloaking Fabric

Table 5.3 shows the cost of running every application in terms of the number of pre-processed elements(bits, triples) used and the rounds of communication for each application. The running time of applications can be directly attributed to the number of rounds of communication. There is still potential to improve the algorithms via batching to reduce the number of rounds of communication. For comparison, Scale-Mamba implements comparison with just 7 rounds of communication compared to 200 rounds in HoneybadgerMPC. MPC joins the darkside[38] reports that for an auction that has 156 comparisons it takes 4.7 seconds for 2 parties. That is 1092 rounds of communication. The 7 bid auction running on Cloaking Fabric takes 17.47 seconds due the fact that there are 1400 rounds of communication. If we just compare the running times with respect to rounds of communication then the two systems perform comparably. The other difference in time could be attributed to Cloaking Fabric involving double the number of parties and offering a robust failure mode.

5.3.6 Secret-Sharing Performance

Table 5.2 benchmarks the bandwidth and time taken for secret-sharing using HbAVSS. We measure the time from the client node which is sharing the value, this includes the time taken to run `createCell` on the application chaincode. HbAVSS allows for multiple values(k) to be shared simultaneously by running multiple instances of AVSS in different

threads separately. The total bandwidth scales linearly at the dealer as well as the nodes but the time taken reduces for larger values of k.

5.3.7 Overhead of Using Fabric

We measure the overhead of using the blockchain and chaincode programming model compared to just running HoneybadgerMPC in Table 5.3. The overhead is defined as the time taken to interact with the chaincode to create metadata, collect results and awaiting results to be published on the blockchain. To measure the time taken in Cloaking Fabric, we measure the duration between the invocation of the first chaincode endpoint that calls `mpcOp` and the invocation of the endpoint that retrieves the published result. To measure the time taken for just the MPC operation, we measure the time in the HoneybadgerMPC codebase to start and complete the MPC operation. The results show that Cloaking Fabric adds a large overhead compared to the total execution time for applications with low running times, however as the running time increases the overhead becomes negligible compared to the total time due to the fact that collecting results and awaiting the results to be published is generally constant time. The difference in overhead time between Linear Regression running 100 epochs to 1000 epochs could be attributed to the more number of blocks created due the client constantly polling the chaincode to collect the results. The created blocks need to be gossiped and committed which takes time.

5.3.8 Performance with Failures

In Table 5.5 we measure the application run times when there is a crashed node. We crash one of the nodes during MPC operation by killing the docker container and we measure the total time it takes. We observe that the running time drops compared to the case with failures. This can be attributed to lower bandwidth the nodes have to deal with. This is a common occurrence in BFT protocols as well and was discussed in BFT under fire[24].

CHAPTER 6: CONCLUSION AND FUTURE WORK

We presented a novel approach for a blockchain system to support private data and allow arbitrary computations on the private data using Secure Multi-Party Computation. From the evaluation above it is evident that compared to other systems like Calypso which support private data on the blockchain, Cloaking Fabric leverages the chaincode (smart contract) programming model and extends it to support MPC functions. The addition of having a MPC coupled with blockchain is not very expensive and scales almost in a constant rate with increase in overall running time.

6.1 FUTURE WORK

1. Testing and benchmarking on large number of nodes : Currently, we restrict our testing to 4 nodes due to the lack of easy deployment methods of Fabric peers. Theoretically fabric supports a large number of nodes in deployment and HoneybadgerMPC is also known to support 100 nodes. It would be interesting to verify in practice and observe performance with scalability.

2. Support for Zero Knowledge Proofs : Applications like linear regression make use of Fixed Point fields. It would be beneficial to have support for Zero-Knowledge ranged proofs to validate the input range entered by the client.

3. Integrating with offline phase of HoneyBadgerMPC : HoneyBadgerMPC has a robust offline phase that generates pre-processing elements on the fly and replenishes their supply on the go. Currently, we resort to using files generated prior to the execution of the MPC application.

4. Improving the efficiency of applications presented : The applications presented here like the auction and linear regression applications can be optimized to reduce the number of rounds of communication. This would make the application run time closer to that of Scale-Mamba.

REFERENCES

[1] P. DuBois and M. Foreword By-Widenius, *MySQL*. New riders publishing, 1999.

[2] M. Stonebraker and L. A. Rowe, *The design of Postgres*. ACM, 1986, vol. 15, no. 2.

[3] J. T. Mentzer, W. DeWitt, J. S. Keebler, S. Min, N. W. Nix, C. D. Smith, and Z. G. Zacharia, "Defining supply chain management," *Journal of Business logistics*, vol. 22, no. 2, pp. 1–25, 2001.

[4] T. Guardian, *50 million Facebook profiles harvested for Cambridge Analytica in major data breach*, 2018 (accessed March 3, 2019). [Online]. Available: https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election

[5] T. Guardian, *CA on 2016 presedential elections*, 2018 (accessed March 3, 2019). [Online]. Available: https://www.theguardian.com/technology/2017/oct/26/cambridge-analytica-used-data-from-facebook-and-politico-to-help-trump

[6] Wired, *CA on 2016 presedential elections*, 2018 (accessed March 3, 2019). [Online]. Available: https://www.wired.com/story/what-did-cambridge-analytica-really-do-for-trumps-campaign

[7] E. Union, *gdpr*, 2018 (accessed March 3, 2019). [Online]. Available: https://eugdpr.org/

[8] E. Union, *gdpr-recital83*, 2018 (accessed March 3, 2019). [Online]. Available: http://www.privacy-regulation.eu/en/recital-83-GDPR.htm

[9] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[10] G. Rapier, "From yelp reviews to mango shipments: Ibm's ceo on how blockchain will change the world," *Business Insider*, vol. 21, p. 2017, 2017.

[11] Y. Luo, H. Jin, and P. Li, "A blockchain future for secure clinical data sharing: A position paper," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. ACM, 2019, pp. 23–27.

[12] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *2016 2nd International Conference on Open and Big Data (OBD)*. IEEE, 2016, pp. 25–30.

[13] A. Kovach and G. Ronai, "Mymedis: a new medical data storage and access system," *None*, 2018.

[14] A. Tapscott and D. Tapscott, "How blockchain is changing finance," *Harvard Business Review*, vol. 1, no. 9, 2017.

[15] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich et al., "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 30.

[16] Hipa, *Access logs*, 2018 (accessed March 3, 2019). [Online]. Available: https://www.hipaaformsps.com/hipaa-access-logs-audits/

[17] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[18] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Advances in Cryptology — CRYPTO 2000*, M. Bellare, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 36–54.

[19] E. Kokoris-Kogias, E. C. Alp, S. D. Siby, N. Gailly, L. Gasser, P. Jovanovic, E. Syta, and B. Ford, "Calypso: Auditable sharing of private data over blockchains," Cryptology ePrint Archive, Report 2018/209, Tech. Rep., 2018.

[20] F. Benhamouda, S. Halevi, and T. Halevi, "Supporting private data on hyperledger fabric with secure multiparty computation," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 357–363.

[21] C. on medium, *Fabric is widely deployed*, 2018 (accessed March 3, 2019). [Online]. Available: https://medium.com/coinmonks/hyperledger-projects-in-real-companies-35016745362c

[22] D. S. Lab and IC3, *HoneybadgerMPC*, 2018 (accessed March 3, 2019). [Online]. Available: https://github.com/initc3/honeybadgermpc

[23] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[24] A. Singh, T. Das, P. Maniatis, P. Druschel, and T. Roscoe, "Bft protocols under fire." in *NSDI*, vol. 8, 2008, pp. 189–204.

[25] L. Lamport et al., "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[26] J. Sousa, E. Alchieri, and A. Bessani, "State machine replication for the masses with bft-smart," *Google Scholar*, 2013.

[27] J. Kwon, "Tendermint: Consensus without mining," *Draft v. 0.6, fall*, 2014.

[28] Chain, *Chain*, 2016 (accessed March 3, 2019). [Online]. Available: https://chain.com/

[29] L. S. Sankar, M. Sindhu, and M. Sethumadhavan, "Survey of consensus protocols on blockchain applications," in *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE, 2017, pp. 1–5.

[30] M. Castro, B. Liskov et al., "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[31] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[32] A. Kate, A. Miller, and T. Yurek, "Brief note: Asynchronous verifiable secret sharing with optimal resilience and linear amortized overhead," *arXiv preprint arXiv:1902.06095*, 2019.

[33] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 31–42.

[34] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Annual International Cryptology Conference*. Springer, 1991, pp. 420–432.

[35] E. Cecchetti, F. Zhang, Y. Ji, A. Kosba, A. Juels, and E. Shi, "Solidus: Confidential distributed ledger transactions via pvorm," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 701–717.

[36] G. Zyskind, O. Nathan, and A. Pentland, "Enigma: Decentralized computation platform with guaranteed privacy," *arXiv preprint arXiv:1506.03471*, 2015.

[37] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits," in *European Symposium on Research in Computer Security*. Springer, 2013, pp. 1–18.

[38] J. Cartlidge, N. P. Smart, and Y. T. Alaoui, "Mpc joins the dark side," *arxiv*, 2018.

[39] E.-O. Blass and F. Kerschbaum, "Strain: A secure auction for blockchains," in *European Symposium on Research in Computer Security*. Springer, 2018, pp. 87–110.

[40] F. Benhamouda, S. Halevi, and T. Halevi, "Supporting private data on hyperledger fabric with secure multiparty computation," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, April 2018, pp. 357–363.

[41] A. Barak, M. Hirt, L. Koskas, and Y. Lindell, "An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 695–712.

[42] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, pp. 305–319.

[43] J. Kreps, N. Narkhede, J. Rao et al., "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.

[44] M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti, "Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric," *arXiv preprint arXiv:1805.08541*, 2018.

[45] S. Beurgel, *Rock Paper Scissors in Solidity*, 2016 (accessed March 3, 2019). [Online]. Available: https://github.com/SCBuergel/ethereum-rps/blob/master/rps-advanced.sol

[46] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter et al., "Secure multiparty computation goes live," in *International Conference on Financial Cryptography and Data Security*. Springer, 2009, pp. 325–343.

[47] O. Catrina and S. de Hoogh, "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks*, J. A. Garay and R. De Prisco, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 182–199.

[48] M. Backes, C. Hritcu, and M. Maffei, "Type-checking zero-knowledge," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 357–370.

[49] G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012, vol. 329.

[50] O. Catrina and S. De Hoogh, "Improved primitives for secure multiparty integer computation," in *International Conference on Security and Cryptography for Networks*. Springer, 2010, pp. 182–199.

[51] D. S. Lab, *Linear Regression*, 2018 (accessed March 3, 2019). [Online]. Available: https://github.com/hybridNeo/HoneyBadgerMPC/blob/dev/honeybadgermpc

[52] F. Samples, *Fabric Samples*, 2018 (accessed March 3, 2019). [Online]. Available: https://github.com/hyperledger/fabric-samples