On Security and Privacy of Consensus-based Protocols in Blockchain and Smart Grid

Inaugural-Dissertation
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften

der Universität Mannheim

vorgelegt von

M.Sc. Avikarsha Mandal
aus Kalkutta (Indien)

Mannheim, 2020

Dekan: Dr. Bernd Lübcke, Universität Mannheim
Referent: Prof. Dr. Frederik Armknecht, Universität Mannheim
Korreferent: Prof. Dr. Erik Zenner, Hochschule Offenburg

Tag der mündlichen Prüfung: 22. Juni 2020

Abstract

In recent times, distributed consensus protocols have received widespread attention in the area of blockchain and smart grid. Consensus algorithms aim to solve an agreement problem among a set of nodes in a distributed environment. Participants in a blockchain use consensus algorithms to agree on data blocks containing an ordered set of transactions. Similarly, agents in the smart grid employ consensus to agree on specific values (e.g., energy output, market-clearing price, control parameters) in distributed energy management protocols.

This thesis focuses on the security and privacy aspects of a few popular consensus-based protocols in blockchain and smart grid. In the blockchain area, we analyze the consensus protocol of one of the most popular payment systems: *Ripple*. We show how the parameters chosen by the Ripple designers do not prevent the occurrence of forks in the system. Furthermore, we provide the conditions to prevent any fork in the Ripple network. In the smart grid area, we discuss the privacy issues in the *Economic Dispatch* (ED) optimization problem and some of its recent solutions using distributed consensus-based approaches. We analyze two state of the art consensus-based ED protocols from Yang *et al.* (2013) and Binetti *et al.* (2014). We show how these protocols leak private information about the participants. We propose privacy-preserving versions of these consensus-based ED protocols. In some cases, we also improve upon the communication cost.

Zusammenfassung

Neuerdings haben verteilte Konsensprotokolle im Bereich Blockchain und Smart Grid große Aufmerksamkeit erhalten. Konsensalgorithmen haben das Ziel, ein Übereinstimmungsproblem zwischen einer Gruppe von Knoten in einer verteilten Umgebung zu lösen. Teilnehmer an einer Blockchain verwenden Konsensalgorithmen, um sich auf Datenblöcke zu einigen, die aus einer geordneten Menge von Transaktionen bestehen. In ähnlicher Weise verwenden Teilnehmer am Smart Grid einen Konsens, um bestimmte Werte (z. B. Energieausbeute, Markträumungspreis, Steuerungsparameter) in Protokollen für das verteilte Energiemanagement zu vereinbaren.

Diese Arbeit fokusiert die Sicherheits- und Datenschutzaspekte einiger gängiger konsensbasierter Protokolle im Bereich Blockchain und Smart Grid. Im Blockchain-Bereich analysieren wir das Konsensprotokoll eines der gängigsten Zahlungssysteme: *Ripple*. Wir zeigen, wie die von Ripple-Designern gewählten Parameter das Auftreten von Gabelungen im System nicht verhindern. Darüber hinaus definieren wir die Voraussetzungen, um Gabelungen im System zu verhindern. Im Bereich Smart Grid diskutieren wir die Datenschutzaspekte des Optimierungsproblems *Economic Dispatch* (ED) und einige seiner neusten Lösungen unter Verwendung verteilter konsensbasierter Ansätze. Wir analysieren zwei State of the Art konsensbasierte ED-Protokolle von Yang *et al.* (2013) und Binetti *et al.* (2014). Wir zeigen, wie diese Protokolle private Informationen über die Teilnehmer preisgeben. Wir schlagen Versionen dieser konsenbasierten ED-Protokolle vor, welche die Probleme hinsichtlich der Preisgabe persönlicher Daten lösen. In einigen Fällen verbessern wir auch die Kommunikationskosten.

Acknowledgements

This doctoral thesis would not be possible without the contribution, guidance, and inspiration of several people. First and foremost, I would like to thank my two doctoral supervisors, Prof. Dr. Erik Zenner and Prof. Dr. Frederik Armknecht. I am deeply grateful to both of them for accepting me as a doctoral student under their supervision and giving me the opportunity to grow as a computer scientist. I am incredibly fortunate to have both of them as my supervisor.

I cannot thank Erik enough for his tremendous support and guidance throughout these years. He always had kept an open-door policy to discuss any problems at any time, gave me the freedom to work on my research questions, provided me regular feedback on any challenges I faced, be it technical or personal, and shared invaluable inputs during my time in Offenburg.

I want to offer my deepest gratitude to Frederik. Whenever any technical roadblocks came into sight in my work, he always helped me figure out the issues and guided me to solve them. This work would not have been possible without many meetings that I had at Mannheim with Frederik. I also thank him for his valuable contributions to all of our joint works throughout my doctoral journey.

I want to thank Prof. Dr. Colin Atkinson and Prof. Dr. Matthias Krause for participating in my defense committee. I am also thankful to the Department of Media and Information Technologies (Offenburg University), School of Business Informatics and Mathematics (University of Mannheim), Group of Theoretical Computer Science and IT Security (University of Mannheim), Graduate School of Offenburg University for giving me an excellent working environment and organizing the doctoral program.

I would like to thank all of my collaborators, particularly Dr. Ghassan Karame. He was the one who introduced me to the exciting areas in blockchain security and suggested focusing on Ripple in my early Ph.D. days. I want to thank all my students whom I worked with over these years, particularly Nuttapol Laoticharoen. His work while visiting Offenburg University showed the practicality of the protocols proposed in this thesis.

Many thanks to my colleagues in the Department of Media and Information Technologies (Offenburg University) for a great time in Offenburg and many bits of help that I received from them. I will miss our hour-long Ph.D. related discussions with Sai Manoj and my office mate Peter, whom I became my friends with outside work. I also thank all my friends from India, Luxembourg, and

especially in Offenburg, with whom I shared a great social life during this time. My gratitude to all my teachers throughout all these years, who helped to build my background and principles to be here where I am today. Also, shout out to my friend Dhruba Ghosh for proofreading my thesis on short notice.

Finally, I am extremely fortunate to have my family on my side with their unconditional support: my parents Baijayanti Baur and Akhil Kumar Mandal, who are with me in all the ups and downs in my life, my brother Avradip (also big congrats to Dia and you, and welcome Aurodyuti to the Mandal family) for inspiring me to pursue an academic career in security and cryptography, my lovely wife Natalie for her incredible patience and keeping me motivated till this date and my beautiful daughter Aalaya for bringing me hope and joy with her smile.

Avikarsha Mandal
Aachen, 2020

Contents

Chapter 1

Introduction

1.1 Motivation

The *consensus* is one of the most fundamental problems in distributed systems research and has been studied over the last four decades. The main goal of the consensus algorithms is to ensure that nodes in a distributed setting can agree on a particular shared state. This area of research has developed over the years since its early theoretical results [1, 2, 3], practical implementations [4], and most recently renewed overwhelming adaptation in *blockchain* [5, 6, 7] and *smart grid* [8, 9].

The *blockchain* or *distributed ledger* stores a growing list of transactions or records in the form of data blocks on top of a *peer to peer* (P2P) network. The blockchain protocols employ consensus such that the participants can agree on the set of transactions to be included in the database. From the security perspective, the consensus protocols in blockchain must be fault-tolerant from malicious (or *byzantine*) attackers. However, the consensus algorithms used by different blockchain protocols are often non-standard. They differ from traditional ones [4] due to practical requirements such as network model, latency, number of participants, etc. Therefore, these new-age blockchain-based consensus protocols require rigorous security assessment and analysis such that claimed security guarantees actually hold in practice.

The *smart grid* is the modern electricity network that supports the information communication channel in parallel to the energy delivery network. It uses a new generation of distributed energy management protocols. The network nodes use consensus algorithms to agree on a set of common parameters such as final electricity output, market-clearing price, estimates of different variables for optimization. The development of consensus protocols for the smart grid is currently an extension of multi-agent control and consensus research [10]. Traditional consensus research [1, 2, 3, 4] focuses on developing fault-tolerant protocols in the presence of byzantine attackers. However, the consensus research in industrial control systems concentrates on asymptotic behavior and

convergence rate of the agreement between nodes. Furthermore, the consensus algorithms in the smart grid often consider several practical factors such as actuation, physical properties, non-linear optimization, network graph [10, 9], etc. In the smart grid, the consensus protocols are iterative. The participants start with some initial values, and asymptotically agree on a specific amount at the end of the protocol. In this agreement process, participants share their inputs with others for computation. However, the data from participants might be privacy-sensitive or confidential. For instance, a set of participants would like to agree on the average energy consumption per month in a locality. Henceforth, if individual nodes send their consumption data to others, it could reveal personal behavioral traces. Therefore, the design of privacy-preserving distributed consensus protocols in a smart grid is necessary and opens a new research direction.

1.2 Contributions

This thesis analyses the security and privacy aspects of some of the new generation consensus protocols in the application area of blockchain and smart grid.

The *Ripple* payment system [11] has evolved as one of the most prominent cryptocurrency and blockchain network. Its consensus algorithm [5] promises much faster ledger closing speed than its competitor Bitcoin's *Proof of Work* (PoW) consensus. In this thesis, our first contribution is on the security analysis of Ripple's consensus protocol. We show that the parameters provided by Ripple designers in the whitepaper [5] do not prevent *blockchain fork*, and might lead to *double spending* in the system. We furthermore present the overlap conditions between consensus participants to prevent any fork in its blockchain.

The *Economic Dispatch* (ED) is a fundamental optimization problem in smart grid energy management. The problem involves the minimization of total operating cost while satisfying some system constraints. As several distributed consensus-based protocols to solve ED problem have been proposed lately to replace traditional centralized calculation, we have observed that most of the current solutions are not secure. In particular, we study two state-of-the-art distributed ED protocols from Yang *et al.* [8] and Binetti *et al.* [9]. In this work, we demonstrate attacks to show how confidential information about participants is leaking while running both protocols and developed two privacy-preserving versions. The first one is called as *Privacy Preserving Economic Dispatch* (*PPED*) protocol [12], which is constructed on top of Yang *et al.* ED protocol [8]. We have analyzed the security of PPED in the information-theoretic setting. In our solution, we also improve upon the communication cost from the original Yang *et al.* protocol. The second protocol is known as *Privacy Preserving Binetti* (*PPB*) [13] based on Binetti's ED protocol [9]. To construct such private ED protocols, we have used cryptographic building blocks from *secure multiparty computation* (SMC) [14, 15].

The PPED protocol considers quadratic cost function, whereas the PPB protocol can be applied to more realistic non-convex cost function optimization

in a smart grid. As the participants in a smart grid are identifiable and regulated, we assume *semi-honest* attacker nodes instead of *byzantine* nodes in the analysis of PPED and PPB protocols.

1.3 Thesis Structure

This thesis is organized into three parts, as follows:

- Part I consists of chapter 2, where we formally define various notions related to consensus, multiparty computation, etc., which will be useful in the latter part of the thesis.
- Part II focuses on the security of the Ripple payment system, particularly on its consensus protocol. Chapter 3 gives a brief introduction to blockchain-related concepts and different blockchain consensus algorithms. Chapter 4 overviews the Ripple payment system and surveys its related security and privacy results. Chapter 5 describes Ripple's consensus protocol and our analysis.
- Part III presents privacy in consensus-based distributed economic dispatch protocols in the smart grid. First, we give the background on the smart grid, its security and privacy challenges, privacy-enhancing technologies in chapter 6. In chapter 7, the ED problem is introduced, and we survey previous works related to privacy in ED protocols. Chapter 8 presents an attack on existing consensus-based ED protocol from Yang *et al.* [8] and introduces the PPED protocol to solve the distributed ED problem in a privacy-preserving manner. Chapter 9 discuses consensus-based ED protocol from Binetti *et al.* [9], we show how it leaks private information and we transform it into a privacy-preserving distributed protocol named PPB. Finally, we conclude our thesis in chapter 10.

Publications: The contents of this thesis are primarily based on three papers and one poster, previously published in TRUST'15 [16], FNSS'16 [12], NordSec'18 [13] and EuroS&P'17 [17]. Some other papers written during the doctoral research but not included in this thesis are [18, 19, 20].

- Chapter 4 and 5 of Part II extend the description and analysis of Ripple previously published in [16].
- Chapter 7 of Part III extends the description and related works of ED previously published in [12] and [13].
- Chapter 8 of Part III extends the results published in [12].
- Chapter 9 of Part III is based on [13].
- The attack discussed in chapter 9 of Part III was first presented in [17].

Part I

Preliminaries

Chapter 2

Preliminaries

This chapter introduces the necessary technical background to understand this thesis. First, we discuss consensus in distributed systems and some well-known results. Then we give some mathematical basis and present underlying concepts of the cryptographic building blocks used in this thesis. We use the terms "node", "party", and "participant" indistinctly.

2.1 What is Consensus?

The *consensus* is a fundamental problem in distributed systems that solves the system reliability in the presence of faulty nodes. At the core, it is an agreement problem where all non-faulty nodes have to agree on a specific value after some nodes propose some value. In distributed systems research, this problem has a long history and has been studied for the last 40 years [1, 2, 3, 4]. The goal of any consensus algorithm is to reach identical decisions. We present the properties of a consensus algorithm as given in [21, p. 150][22, p. 18] as follows:

Definition 2.1.1. *(Consensus)* In a n node system, suppose t nodes are faulty and every node P_i has some input value x_i. The consensus holds if the following properties are satisfied:

- **Agreement:** All non-faulty nodes agree on the same value.
- **Validity:** The agreed value is one of the input values possessed by the nodes.
- **Termination:** All non-faulty nodes should terminate within a finite time.

The security of any consensus algorithm is typically evaluated with *property-based* approach, i.e., showing how these properties such as agreement, validity, termination are satisfied. The quality of a consensus protocol depends on different measures such as maximum number of faulty nodes t (in terms of n) a protocol can tolerate, worst case termination time of honest nodes, and the communication complexity.

2.1.1 Byzantine General Problem

The *byzantine general problem* [1, 2] was introduced by Lamport, Pease, and Shostak, and is a fundamental problem in fault-tolerant distributed systems. This problem states an agreement problem among a set of nodes("generals") in the presence of faulty nodes (known as *byzantine*). A node is called *byzantine* if it can have arbitrary behavior such as not sending any messages, sending wrong messages to different nodes, lying about input. Byzantine nodes can collude together or can be controlled by one specific adversary. We call a system reached *byzantine agreement* when the nodes reach consensus as defined in 2.1.1 in the presence of byzantine nodes. Lamport *et al.* in [2] showed that the necessary and sufficient condition to reach a byzantine agreement is $t < \frac{n}{3}$. In the cryptographic sense, byzantine attackers are the strongest possible attackers, also known as malicious attackers. The genre of protocols that solves consensus with byzantine faulty nodes are known as *byzantine fault tolerant* (BFT) protocols.

2.1.2 Models for Communication in Distributed Setting

There are different network models for communication to design distributed consensus algorithms:

Synchronous Model

In this model, nodes function in synchronous rounds. In each round, each node can send a message to the other nodes, receive messages from the other nodes, and perform some local computation. More specifically, there exist a known fixed upper bound δ for the time to send one message from one node to another and a known fixed upper bound ϕ on the relative computational speeds of different nodes [23]. In general, the synchronous model assumes *point to point* communication channel such that nodes are connected with pairwise secure and authenticated channels. Furthermore, a fully connected graph is a standard setting for synchronous consensus protocols [2].

Partial Synchrony Model

Some models consider a relaxed setting to design consensus protocols in practice. Dwork *et al.* in [23] introduced *partial synchrony model* where there exist fixed bounds for message delivery (δ) and relative computation speed (ϕ). However, δ and ϕ are not known beforehand by the protocol participants. Another variant is known as the *eventual synchrony model*, where synchrony eventually holds after some unknown but fixed time interval.

Asynchronous Model

In the asynchronous network model, messages arrive after finite but unbounded time. This model is also known as the *eventual delivery* model. In a fully

asynchronous model, messages might be arbitrary dropped or delayed. More specifically, there are no upper bounds for δ and ϕ.

2.1.3 Fischer-Lynch-Paterson (FLP) Impossibility Result

Fischer, Lynch, and Paterson in [3] proved that no deterministic algorithm could solve the consensus problem in a fully asynchronous network model even with one faulty byzantine node. This influential result in distributed consensus is known as *FLP impossibility result*[1]. In the eventual delivery model, even though there exists no deterministic algorithm to reach the consensus, consensus protocol construction is possible with randomization [3].

For further reading about consensus problems in distributed systems, we recommend the book by Lynch [21].

2.2 Mathematical and Cryptographic Preliminaries

We describe some basic notions from graph theory and algebraic structures used in this thesis in brief.

- A group G is a set and an associated binary operation $\cdot$ that takes two elements of the set and maps the elements to a third element. The operation satisfies four group axioms.
  - Closure: $a, b \in G \Rightarrow a \cdot b \in G$
  - Associativity: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
  - Identity: There exists $e \in G$, such that for all $a \in G$ we have $a \cdot e = e \cdot a = a$
  - Inverse: For all $a \in G$, there exists $a^{-1} \in G$, such that $a \cdot a^{-1} = a^{-1} \cdot a = e$, where e is the identity element from the previous condition.
- A group G is cyclic, if there exists a generator element $g \in G$, such that any other element of G can be generated by repeated application of g with itself. Equivalently, $g \in G$ is generator of cyclic group G if for any $h \in G$ there exists an integer i such that $h = g^i$.
- A group G is commutative if for all $a, b \in G$, we have $a \cdot b = b \cdot a$.
- A field F is a set, associated with two binary operations, addition $+$ and multiplication $\cdot$, such that:
  - it is a commutative group over addition $+$.
  - the additive identity is zero (0). There exists a different multiplicative identity $1 \neq 0$.

[1] This paper was awarded Dijkstra Prize in 2001

  - $G \setminus \{0\}$ is a commutative group over multiplication operator $\cdot$.
  - The multiplication operator $\cdot$ distributes over the addition operator $+$. That is for all $a, b, c \in G$, we have $a \cdot (b + c) = a \cdot b + a \cdot c$.

- A field with finite number of elements is called *Finite field* or *Galois Field*. The number of elements in a Finite Field is always p^k, for some prime p and non negative integer k. Finite fields are called prime field, if the total number of elements is a prime p. All prime fields of size p are isomorphic to set of natural numbers modulo p. This field is denoted as F_p or $\mathbb{Z}_p$.

- A graph consists of a set of nodes V and a set of directed edges $E \subseteq V \times V$. In case of undirected graph, E consists of unordered tuples; in case of directed graph E consists of ordered tuples. A graph (V, E) is connected, if for any $u, v \in V$, there exists a sequence of nodes $u_1, u_2, \cdots, u_k$ such that $(u, u_1), (u_2, u_3), \cdots, (u_k, v) \in E$. A graph (V, E) is fully connected if for any $u, v \in V$, $(u, v) \in E$.

- A *hash function* $H : \{0,1\}^* \to \{0,1\}^h$ maps an arbitrary length string to a short digest. Typically h is about 128 or 256. For a regular hash function, the expected property is the output should be random for any input. However, cryptographic hash functions exhibit additional properties.

  - Pre-image Resistance: For any polynomial time adversary, given randomly chosen $y \in \{0,1\}^h$ it is hard to output any $x \in \{0,1\}^*$, such that $H(x) = y$.
  - Collision Resistance: For any polynomial time adversary it is hard to output any (x_1, x_2), such that $H(x_1) = H(x_2)$.
  - One-wayness: For any polynomial time adversary, for any randomly chosen input string x of some length, given $H(x)$ it is hard to output x.

- A *digital signature* scheme is a public key cryptographic primitive consisting of three algorithms Key Generation, Signing, and Verification. The Key generation algorithm generates a public key and a corresponding private key. Any party holding the private key can *sign* on arbitrary messages. Any other party having access to the public key can *verify* the authenticity of message signature pairs.

2.3 Secure Multiparty Computation (SMC)

Secure multiparty computation (known as SMC or MPC or SMPC) is a family of cryptographic techniques for privacy-preserving computation. The goal of SMC is to enable multiple parties to jointly compute a function while keeping input data private. The SMC protocols exist in two-party as well as multiparty setting. One classic example of SMC is a solution to *millionaires problem* by Yao's garbled circuit construction [24], where two millionaires find who has more

wealth without revealing their wealth. Since then, many SMC protocols have been proposed, and existing literature includes [25, 14, 26, 27].

Succinctly, one can define a generic SMC protocol as following [28]:

Definition 2.3.1. *(SMC)* In a n node system, parties $P_1, \cdots, P_n$ want to compute the function $y = f(x_1, \cdots, x_n)$ where x_i is the private input of P_i. Consider an external adversary $\mathcal{A}$ that can corrupt and control a subset of participating parties (minority). An SMC protocol should satisfy the following security properties:

- **Input privacy:** parties learn the output y and the information inferred from y, nothing else can be learned from the protocol execution.
- **Correctness:** all honest parties are guaranteed to learn the correct output y in presence of adversary $\mathcal{A}$.

Let's take an example of the generalized millionaire's problem with n parties where $x_1, \cdots, x_n$ be the wealth of individual parties. Clearly, $f(x_1, \cdots, x_n) = i$ if $x_i > x_j \; \forall \; i \neq j$. An SMC protocol to solve the generalized millionaire's problem should follow the security properties as mentioned earlier in 2.3.1. While running the SMC protocol, only the identity of the richest millionaire is allowed to be revealed to all parties (*input privacy*). Second, the SMC protocol should output the correct result, i.e., the richest party is guaranteed to win, and an adversary $\mathcal{A}$ cannot alter the result (*correctness*).

There are two distinct approaches to construct SMC protocols. The first genre is *Yao's garble circuit* approach, where the function is computed as a binary circuit. The gates of the circuit are "encrypted" to form a garbled circuit. The security of such schemes relies on the computational assumption and follows from the security of the encryption scheme. The second family of protocols is secret sharing based, where the function is presented as an arithmetic circuit. In general, the Yao-based garbled circuit approach is more suitable for two-party computation, and the secret sharing based strategy is ideal for the multi-party setting. In this work, our focus will be on secret sharing based SMC protocols.

The Simulation Paradigm

A more formal definition of SMC [29, 30][31, sec. 7.1] considers *ideal/real simulation paradigm*. In the "ideal world", an incorruptible or trusted party helps the parties to perform the computation. Parties can directly send their input to the trusted party, trusted party computes the function, and send back the output to them. It models the idealized version of the protocol, including any allowable information leakage. In the "real world", the parties run the protocol among themselves to compute the function without any trusted party. The real world SMC protocol is secure if compared to the ideal world execution, the real world protocol execution does not reveal more information to the adversary. In other words, if the adversary learns the same information in both worlds, the real world protocol is secure.

2.3.1 Adversarial Model

As we mentioned earlier in 2.3.1, the model for SMC considers an adversary $\mathcal{A}$ that controls some subset of the participating parties and wants to attack the protocol execution. The controlled subset by the adversary is known as *corrupted* parties. The adversary can be classified based on the corruption strategy, adversarial behavior, and computational power. The following classification is based on [32]:

Corruption Models

The adversary can corrupt the participating nodes in two ways: *static corruption* and *adaptive corruption*. In a static corruption model, the adversary can control and corrupt a fixed number of participants. The role of honest parties and corrupted parties are fixed throughout the computation in this model. In the adaptive corruption model, the adversary has the ability to corrupt parties during the computation. Once a party is corrupted at some point will remain corrupted throughout the computation in this model. Another model known as *proactive corruption* considers corrupted parties to be corrupted for a certain amount of time. A computing party in a proactive corruption model can be corrupted during computation like in adaptive corruption but can be honest again after a specific time.

Adversarial Behavior

The adversarial behavior can be classified based on the action of the corrupted parties during computation. Such as behavior can be classified as *semi-honest* and *malicious*. In semi-honest (also known as *passive* or *honest-but-curious*) adversary model, the corrupted parties strictly follow the protocol specification, but may analyze the message exchanges to gain additional information during the protocol execution. In malicious (also known as "active") adversarial model, the corrupted party can deviate arbitrarily from the protocol specification. This model is a much stronger adversarial model than semi-honest, because the adversary has additional freedom of deviating from the protocol. Furthermore, another model in SMC known as *rational adversary model* considers game-theoretic strategies to model the rational behavior of corrupted parties. This adversary model is a stronger adversary than semi-honest but weaker than the malicious model.

Computational Power

In SMC, the adversarial power is modeled based on two computational complexity categories. First, the adversary is computationally bounded. In this setting, we assume *probabilistic polynomial time* (PPT) adversaries which cannot solve cryptographic hard problems. This is known as *computationally bounded* setting.

The second type of adversary has no computational limits, known as *computationally unbounded* adversary. This type of adversary comes under *information-theoretic setting*. The results in this setting do not rely on any cryptographic assumptions of complexity classes. Any protocol which is secure in the information-theoretic setting is trivially secure in the computationally bounded setting.

2.3.2 SMC Security Guarantee

The security guarantees in SMC protocols can be categorized as following [33, 32, 34]:

Information-theoretic Security

An SMC protocol achieves *information-theoretic security* or *unconditional security* or *perfect security* if the adversary does not obtain any additional information running the real world protocol than it learns under ideal setting (with a trusted third party). The result of a real world execution with a real adversary should be the same as the result of ideal execution with a trusted party and ideal world adversary. This security level is achievable with a computationally unbounded adversary in the information-theoretic setting. In this model, it is usually assumed that the parties are connected with ideal *private channels* where the adversary cannot eavesdrop or modify the message communication between two honest parties.

Statistical Security

The *statistical security* level is quite similar to perfect security. In this security level, the adversary learns no additional information than in an ideal setting but with a negligible probability. The result of a real world protocol execution with a real adversary should be statistically close to the result of ideal execution with a trusted party and ideal world adversary. Once again, it considers computationally unbounded adversary in the information-theoretic setting.

Computational Security

An SMC protocol can achieve *computational security* level against a PPT adversary such if breaking the security of the protocol, implies the adversary has to solve a computationally hard problem. Any functionality can be securely computed under appropriate cryptographic assumptions achieves computational security [35, 36].

2.4 Secret Sharing

Secret sharing is one of the important techniques used in SMC protocols. Informally speaking, it involves distributing a secret among a group of participants

such that the *share* of each party does not reveal anything about the secret, but together they can reconstruct the secret value. The idea of secret sharing came independently from Adi Shamir [37] and Bob Blakley [38] to overcome the single point failure problem of secret data storage. It has been a subject of research by its own with various applications (e.g., SMC, byzantine agreement, threshold cryptography, attribute-based encryption). A detailed survey on different secret sharing mechanisms can be found in [39]. Some scheme needs everyone's shares to reconstruct the secret; on the other hand, some schemes require only a subset of the parties are needed to reconstruct the secret. The latter ones are known as *threshold sharing schemes*. A threshold secret sharing scheme can be described as [40]:

Definition 2.4.1. *((t, n) threshold secret sharing scheme)* A (t, n) threshold secret sharing scheme can take s as a secret input and output n shares guaranteeing two following properties:

- **Recoverability:** Any subset of t shares can be used to reconstruct the secret s.
- **Secrecy:** A subset of less than t shares does learn anything about s.

In our work, we particularly focus on Shamir's secret sharing scheme [37], which is a backbone of BGW protocol [14].

2.4.1 Shamir's Secret Sharing

Shamir's secret sharing scheme [37] is based on polynomials over a finite field $\mathbb{F}$. A (t, n) Shamir's threshold scheme is perfectly secure against a semi-honest adversary controlling $t-1$ nodes. The necessary condition for this scheme is $|\mathbb{F}| > n$, where n is the number of participants. For simplicity, we can consider $\mathbb{F} = \mathbb{Z}_p$ such that the prime p is bigger than n ($p > n$). In real-world applications, the prime p that defines the field size is much bigger than n to avoid overflows. If we want to design a $(2, n)$ threshold secret sharing scheme, the secret could be the slope of a line, and each share can be distinct points on the line. Henceforth, 2 parties can find the slope of the line, but one point on the line says nothing about the secret. Similarly, this idea can be generalized as for $(3, n)$ scheme with a quadratic function and for (t, n) scheme with a $(t-1)$ degree polynomial function. Note, any t points in a two-dimensional plane uniquely determines a polynomial of degree $\leq t-1$ (if such a polynomial exists). Shamir's scheme consists of three steps: i) Initialization, iii) Distribution of Shares, and iii) Reconstruction. In the following, we explain the steps using Shamir's (t, n) threshold secret sharing scheme:
Consider a dealer who wants to share the secret $s \in \mathbb{Z}_p$ among the parties $P_1, \ldots, P_n$.

i) Initialization:

The dealer selects n distinct non-zero elements $x_1, \ldots, x_n$ from $\mathbb{Z}_p$ (public).

ii) Distribution of Shares:

The dealer constructs a random polynomial $f_s(x) \in \mathbb{Z}_p[X]$ of degree at most $t-1$ such that $f_s(0) = s$. This can be done by choosing uniformly random $t-1$ elements from $\mathbb{Z}_p$ as $a_1, \ldots, a_{t-1}$ and defining $f_s(x)$ as follows:

$$f_s(x) = s + a_1 x + \cdots + a_{t-1} x^{t-1} \mod p \tag{2.1}$$

Thereafter, the dealer can compute $y_i = f_s(x_i)$ (for $1 \leq i \leq n$) and distribute the share y_i to P_i. As a result, every party P_i gets the a point (x_i, y_i) on the polynomial $f_s(x)$ as a secret share.

iii) Reconstruction:

The secret reconstruction can be done with polynomial interpolation. If t distinct points are known from t parties, the polynomial (degree $\leq t-1$) can be constructed with some interpolation methods (e.g., vandermonde matrix, lagrange interpolation). One can use *lagrange interpolation* method for reconstruction of the secret as solving a system of linear equation with vandermonde matrix is costlier. Without the loss of generality, we suppose that the shares use for reconstruction are $y_1, \ldots, y_t$. The polynomial $f_s(x)$ can be represented as follows:

$$f_s(x) = \sum_{i=1}^{t} y_i \cdot \delta_i(x) \tag{2.2}$$

Here, $\delta_1(x), \ldots, \delta_t(x)$ are lagrange basis polynomials of degree at most $t-1$. Let's take an example of $\delta_1(x)$ polynomial, it has at most $t-1$ roots as $x_2, \cdots, x_t$ and $\delta_1(x_1) = 1$ (as $f_s(x_1) = y_1$). This polynomial can be represented as: $\delta_1(x) = C_1 \cdot (x-x_2) \cdot (x-x_3) \cdots (x-x_t)$ where C_1 is a constant. Now, the constant term C_1 can be found as $C_1 = \frac{\delta_1(x_1)}{(x_1-x_2)\cdot(x_1-x_3)\cdots(x_1-x_t)} = \frac{1}{(x_1-x_2)\cdot(x_1-x_3)\cdots(x_1-x_t)}$. Henceforth, $\delta_1(x) = \frac{(x-x_2)\cdot(x-x_3)\cdots(x-x_t)}{(x_1-x_2)\cdot(x_1-x_3)\cdots(x_1-x_t)}$ and similarly one can define $\delta_i(x)$ as:

$$\delta_i(x) = \prod_{j=1, j\neq i}^{t} \frac{x - x_j}{x_i - x_j} \tag{2.3}$$

Recoverability: Let's see how any t shares can recover secret s. First, one can find:

$$\delta_i(0) = \prod_{j=1, j\neq i}^{t} \frac{-x_j}{x_i - x_j}$$

This $\delta_i(0)$ is a constant depends on which share holders are involved but independent of the shares y_i's. This value can be pre-computed and as we have the shares $y_1, \ldots, y_t$'s, the secret can be found as: $s = f_s(0) = \sum_{i=1}^{t} y_i \cdot \delta_i(0)$.

Secrecy: Here we verify whether Shamir's scheme satisfies the secrecy property i.e. less than t shares reveal anything about the secret s or not. Without the loss of generality, suppose $t-1$ parties $P_2, \ldots, P_t$ are contributing

their respective shares $y_2, \ldots, y_t$ to reconstruct the secret. We need to show these $t-1$ shares do not reveal any information about the secret s. The secret s can be written as: $s = f_s(0) = \sum_{i=1}^{t} y_i \cdot \delta_i(0)$. This implies $y_1 \cdot \delta_1(0) = s - \sum_{i=2}^{t} y_i \cdot \delta_i(0)$. Note that $\delta_1(0) = \prod_{j=2}^{t} \frac{-x_j}{x_1 - x_j}$, is non-zero as all x_j and $x_1 - x_j$'s are non-zero. Hence it follows that without knowing the value of y_1, the secret value s remains unknown. Perfect security is achieved as we have a bijective relation between any possible value of $s \in \mathbb{Z}_p$ and any possible value of the missing share $y_1 \in \mathbb{Z}_p$.

Example:

Let's assume a dealer wants to distribute a secret among three parties P_1, P_2, P_3 using $(2, 3)$ Shamir's scheme i.e., can tolerate upto one corrupt party. The dealer chooses to work in the field $\mathbb{F} = \mathbb{Z}_{13}$ and share the secret $s = 7$. He picks $a_1 \in \mathbb{F}$ uniformly at random. Suppose $a_1 = 5$ and he constructs the polynomial:

$$f_s(x) = s + 5x \mod 13$$

Now the dealer can compute the shares $y_1 = f_s(1) = 7 + 5 \mod 13 = 12$, $y_2 = f_s(2) = 7 + 10 \mod 13 = 4$ and $y_3 = f_s(3) = 7 + 15 \mod 13 = 9$. Each share y_i is sent privately to respective party P_i. Let's assume P_1 and P_3 together want to reconstruct the secret from their share. We can use lagrange method as explained previously. We use equation 2.3 to get:
$\delta_1(x) = \prod_{j=3} \frac{x-x_j}{x_1-x_j} \mod 13 = \frac{x-3}{1-3} \mod 13 = (x-3)(1-3)^{-1} \mod 13 = (x-3)(-2)^{-1} \mod 13 = (x-3)(11)^{-1} \mod 13 = (x-3)\cdot 6 \mod 13 = (6x-18) \mod 13$. Similarly, we can find $\delta_3(x) = \prod_{j=1} \frac{x-x_j}{x_3-x_j} \mod 13 = \frac{x-1}{3-1} \mod 13 = (x-1)(2)^{-1} \mod 13 = (x-1)\cdot 7 \mod 13 = 7x - 7 \mod 13$. Finally, we can find the secret s by using equation 2.2 as:
$s = f_s(0) = \sum_{i \in 1,3} y_i \cdot \delta_i(0) = y_1 \cdot \delta_1(0) + y_3 \cdot \delta_3(0) = 12 \cdot (6 \cdot 0 - 18) + 9 \cdot (7 \cdot 0 - 7) \mod 13 = (-216 - 63) \mod 13 = -279 \mod 13 = 7$.

2.5 Secure Computation from Secret Sharing

Ben-Or, Goldwasser and Widgerson (BGW) protocol [14] is a foundational SMC protocol in information-theoretic model. They demonstrated that any function with n-ary input can be computed with *perfect* (information-theoretic) security, assuming private encrypted channel. The main results of their paper were as follows considering a complete synchronous network of n parties with pairwise private encrypted channel:

Theorem 1. *For every n-ary input function f, there exits a protocol to compute f with perfect security in presence of τ semi-honest adversaries as long as $\tau < \frac{n}{2}$.*

Theorem 2. *For every n-ary input function f, there exits a protocol to compute f with perfect security in presence of t malicious adversaries (or Byzantine) as long as $\tau < \frac{n}{3}$ (requires a broadcast channel).*

In the following, we describe how addition and multiplication gates are securely evaluated. The idea can be extended to arbitrary functions, as any arbitrary function can be expressed in terms of addition and multiplication gates.

2.5.1 Secure Addition

A secure addition protocol can be implemented using the homomorphic property of additive secret sharing [41]. Suppose the set of participants of the multiparty protocol hold shares of a value $x \in \mathbb{F}_p$. The ordered set of shares is denoted by $[x]_p$, the order denotes which share is owned by which participant. If the shares are generated by using a degree t polynomial f, then the set of shares is also denoted as $[x; f]_p$. As the secret sharing scheme is linear, the following properties hold for any $x, y, \alpha \in \mathbb{F}_p$ and degree t polynomials $f, g : \mathbb{F}_p \to \mathbb{F}_p$

- $[x; f]_p + [y; g]_p = [x + y; (f + g)]_p$
- $\alpha[x; f]_p = [\alpha x; f]_p$
- $[x; f]_p \cdot [y; g]_p = [xy; (fg)]_p$

Here, share addition $(+)$ and share multiplications $(\cdot)$ are defined as follows. Suppose, $[x]_p = (x_1, x_2, \cdots, x_n)$ and $[y]_p = (y_1, y_2, \cdots, y_n)$, then

- $[x]_p + [y]_p = (x_1 + y_1, x_2 + y_2, \cdots, x_n + y_n)$
- $\alpha[x]_p = (\alpha x_1, \alpha x_2, \cdots, \alpha x_n)$
- $[x]_p \cdot [y]_p = (x_1 y_1, x_2 y_2, \cdots, x_n y_n)$

Because of the linear homomorphic property of the secret sharing scheme, the secure addition protocol can be trivially realized where parties generate $[x+y]_p$ by calculating $[x]_p + [y]_p$. This is a non interactive protocol with perfect security.

2.5.2 Secure Multiplication

As described in the previous section, $[x]_p \cdot [y]_p$ which can be calculated locally is in fact $[xy]_p$. However, the polynomial corresponding to the resultant share is a degree $2t$ polynomial, even though the original polynomial is of degree t. Genaro, Rabin and Rabin [42] described a secure multiplication protocol to address this issue.

Suppose, we have sets of shares $[x; f]_p$ and $[y; g]_p$ which have been distributed to a set of players $P_1, \cdots, P_n$. As we have discussed before, $[xy; (fg)]$ can be calculated locally by individual players. We know,

$$xy = (fg)(0) = \sum_{i=1}^{n} \lambda_i (fg)(i) \mod p.$$

Here λ_i's (for $1 \leq i \leq n$) are known Lagrange multipliers,

$$\lambda_i = \prod_{\substack{1 \leq k \leq n \\ k \neq i}} \frac{k}{k-i} \mod p.$$

Every player P_i can share their share of xy (i.e. $(fg)(i)$) using a degree t polynomial h_i. i.e. player P_i can choose a random degree polynomial h_i s.t. $h_i(0) = (fg)(i)$ and send $h_i(j)$ to every other player P_j. Now every player can locally compute shares of xy with respect to the degree t polynomial

$$H(x) = \sum_{i=i}^{n} \lambda_i h_i(x).$$

This holds because,

1. $H(0) = \sum_{i=1}^{n} \lambda_i h_i(0) = \sum_{i=1}^{n} \lambda_i (fg)(i) = xy$, which implies $(H(1), \cdots, H(n))$ are valid shares of xy.
2. each share $H(i) = \sum \lambda_j h_j(i)$ can be locally computed by player P_i.

Combining secure addition and multiplication we can securely evaluate any arithmetic circuit.

2.5.3 Bit Decomposition and Bitwise Circuit Evaluation

If we want to compare two numbers, the resultant circuit cannot be written as a small constant depth arithmetic circuit. If we write the operation as a polynomial, the resultant circuit will have gates proportional to the field size which can be huge. Suppose, we have access to the bitwise sharings $([a_0]_p, \cdots, [a_{\ell-1}]_p)$, $([b_0]_p, \cdots, [b_{\ell-1}]_p)$. Here, $a_i, b_i \in \{0, 1\} \subset \mathbb{F}_p$ for $i \in [0, \ell-1]$. Then the function $\sum_{i=0}^{\ell-1} a_i 2^i \overset{?}{<} \sum_{i=0}^{\ell-1} b_i 2^i$ can be securely computed in constant rounds.

In [15], Damgård *et al.* presented a novel constant round unconditionally secure bit decomposition protocol. Suppose $[a]_p$ are shares of $a \in \mathbb{F}_p$. The bit decomposition protocol takes $[a]_p$ as input and outputs $([a_0]_p, \cdots, [a_{\ell-1}]_p)$, where $a = \sum_{i=0}^{\ell-1} a_i 2^i$. Using this bit decomposition protocol we can securely evaluate any bitwise circuit e.g. comparison or finding maximum index.

2.6 Summary

We have described the necessary background on consensus, mathematical preliminaries, and secure multiparty computation (SMC) to understand this thesis. In the SMC section, we have focused primarily on secret sharing-based arithmetic circuit protocols.

Part II

On Security of the Ripple Consensus Protocol

Chapter 3

A Brief Introduction to the Blockchain

3.1 Introduction

The term *blockchain* is often used to refer to current distributed ledger technologies. Generally speaking, a blockchain is a distributed database that stores a growing list of transactions or records in the form of a chain of blocks. Each data block is immutable through cryptographic algorithms to ensure the integrity of the transactions. In a blockchain, instead of a central authority, a group of nodes that do not necessarily trust each other, maintain shared ledger states through some distributed protocol. The blockchain stores an ordered set of transactions where distributed notes run a consensus protocol to agree on the contents of the transactions and their order. The information is stored in a sequence of blocks, and individual blocks can contain one or more transactions. The concept of the blockchain originated from Satoshi Nakamoto's Bitcoin whitepaper in 2008 [43] and further open-source deployment in 2009 as a part of Bitcoin software. The Bitcoin is a decentralized electronic payment system and a cryptocurrency which uses blockchain as its public ledger for monetary transaction. It uses the Proof of Work (PoW) based consensus mechanism. The Bitcoin system allows parties to transfer monetary values without any central institution such as a bank. As a central authority cannot verify the validity of a transaction, the distributed network of nodes has to reach a consensus on whether or not a transaction is valid. In Bitcoin PoW, a group of nodes known as *miners* solve a hard computational problem to generate new blocks. Since the inception of bitcoin, many blockchain initiatives such as Ethereum [6], Ripple [11], and Hyperledger [7] have received considerable attention in academia as well as in industry. The current usage of blockchain is not only limited in decentralized cryptocurrency similar to Bitcoin but also applied in smart contracts, supply chains, Internet of Things (IoT), Industry 4.0, smart grid, etc. In this chapter, we will give an overview of blockchain systems and consensus algorithms related to this thesis.

3.2 Blockchain Overview

3.2.1 Definition

The definition of a blockchain can be application-specific, and one can find several definitions with its evolving features and requirements [44, 45]. Cachin and Vukolić in [44] defined a blockchain as:

> "a distributed database holding a continuously growing list of records, controlled by multiple entities that may not trust each other."

The International Organization for Standards (ISO) is currently working on the standards for blockchain terminologies in ISO/TC 307. They have informally described blockchain in [46] as:

> "a shared, immutable ledger that can record transactions across different industries, thus enhancing transparency and reducing transaction costs. It is a digital platform that records and verifies transactions in a transparent and secure way, removing the need for middlemen and increasing trust through its highly transparent nature."

In 2018, NIST released a technical overview of blockchain and informally defined it as [45]:

> "Blockchains are distributed digital ledgers of cryptographically signed transactions that are grouped into blocks. Each block is cryptographically linked to the previous one (making it tamper evident) after validation and undergoing a consensus decision. As new blocks are added, older blocks become more difficult to modify (creating tamper resistance). New blocks are replicated across copies of the ledger within the network, and any conflicts are resolved automatically using established rules."

3.2.2 Key Concepts

We discuss some key concepts in blockchain to get a better understanding.

Blocks

A *block* (also known as a *ledger*) is a data structure that contains a header and a list of transactions in the blockchain. Every block is identifiable with its hash value in the block header. Each block has a hash pointer connected to the previous block, thus creating a blockchain (see Fig. 3.1). A Hash pointer points to the address where the data is stored and includes the hash of that data. The hash pointers link data blocks together, starting from the *genesis block* to make blockchain tamper-resistant. If an adversary wants to tamper a specific block in the blockchain, he needs to change every hash pointer of all preceding blocks

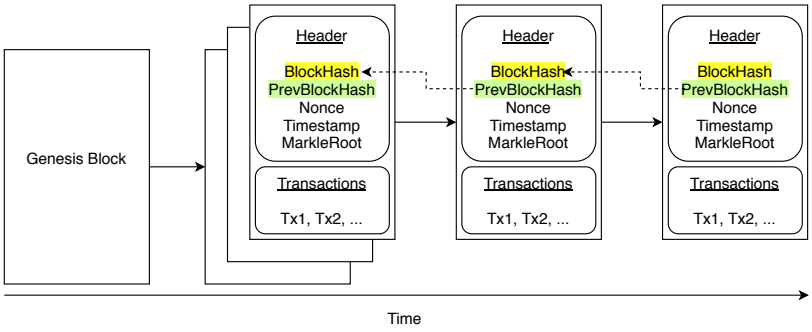

Figure 3.1: Blocks in Blockchain

leading up to the genesis block. Generally, a block header has five attributes, such as block hash value, previous block hash value, nonce, timestamp, and Merkle root. The nonce is a random value found during the mining process in PoW based blockchains. Finding out an appropriate nonce that creates a valid block in PoW blockchain is a hard problem. Miners use the majority of their computation power to find such nonces. The set of transactions in a block uses *Merkle tree* data structure representation. It is a binary hash tree, where at a lower level, every two transaction hashes are grouped into one to make new parent hash and finally converges to the Merkle root hash at the top of the tree. Merkle tree helps to preserve the transaction orders in a block, and one can verify a transaction in a block against the root efficiently in logarithmic time.

Transactions

Transactions are the atomic elements inside a block, particularly in cryptocurrency-based blockchains. A transaction is a transfer of some monetary tokens or coins which is broadcasted and collected in a block. A user needs to sign a transaction with its private key before broadcasting to the network. The transactions are irreversible once they get confirmed in the blockchain. Anyone can see every transaction details inside a block as they are unencrypted. In blockchain, there are two types of transaction models: i) *unspent transaction output (UTXO)*, and ii) *account-based transaction*. Bitcoin and cryptocurrencies based on Bitcoin use UTXO transaction model. The key elements of a UTXO transaction are a set of inputs, outputs, and the transaction hash known as *transaction identifier*. In UTXO, the entire history of a coin transaction is recorded with unspent outputs where each output has an owner and a value. The total monetary value in all inputs must be greater or equal, then the total value of all outputs to produce a valid transaction. Cryptocurrencies like Ripple and Ethereum use account-based transaction model. It is much simpler where all transactions are recorded based on sender accounts. The blockchain records the changes in the user account balance due to a transaction rather than recording the history of a coin movement.

Nodes

A blockchain operates on top of a peer to peer (P2P) network where nodes can have different roles while running a specific blockchain protocol. For example, Bitcoin supports three types of nodes such as full node, miners, and lightweight clients [47]. In general, the whole blockchain is shared across distributed nodes, and every node can have a replicated copy of the blockchain. A full node in the Bitcoin network maintains a complete copy of the blockchain. Full nodes are dedicated to check for new incoming transactions and blocks in the network and forward them to other nodes. They can also validate the transactions inside a new block. The miners are full nodes who are additionally responsible for doing PoW computation to add a new block on top of the existing Bitcoin blockchain. Finally, the lightweight clients use Simple Payment Verification (SPV) protocol to verify a transaction included in the blockchain. These nodes only require to store the block header rather than maintaining the full blockchain. As Ripple blockchain does not employ PoW, it does not have miner nodes like Bitcoin or Ethereum. Instead, a set of nodes called *validators* (also known as *validating nodes*) take part in the consensus process to validate and add new blocks in the blockchain. Other than the nodes, a *network user* is a person or an entity who uses the blockchain network such as making or receiving a transaction.

Digital Signatures

Digital signatures play an important role in blockchain while sending a transaction or a block. Transactions or blocks are hashed and digitally signed by the sender before broadcasting to the other nodes for data integrity and authenticity. The digital signature algorithms have three steps: i) *key generation*, ii) *signing*, and iii) *verification*. In the key generation, anyone can create a private key and a public key. In signing, the sender signs data such as transaction with its private key and broadcasts the transaction with the signature to other nodes. In verification, other nodes can verify the authenticity of the transaction using the signature, the transaction, and the public key of the sender. In a centralized system, a Public Key Infrastructure (PKI) is required to bind the user identity with its public key. Cryptocurrency-based blockchains (e.g., Bitcoin, Ripple, Ethereum) use public keys as pseudonyms instead of PKI. Blockchain users can generate as many key pairs by themselves, and the hash of the public key is known as the user address. Some blockchain protocols include some fees to create a new address to prevent Sybil attacks. In Sybil attacks, the attackers create a large amount of pseudonyms to hamper reputation of a network. Most of the blockchain protocols use *elliptic curve digital signature algorithm* (ECDSA) over the secp256k1 curve, which provides 128-bit security.

Blockchain Consensus Mechanism

In a centralized banking system, a trusted central authority controls the validity of a specific transaction. The central authority has access to control privileges and can take the necessary measures against attacks such as double-spending.

A decentralized blockchain replaces these centralized trust systems with a consensus mechanism. When a new transaction seeks validation, each validating nodes can add or reject that transaction in its *candidate block* locally on top of the global ledger. In the consensus phase, the nodes communicate with each other, and the majority reach an agreement on the next candidate block to be added to the global blockchain. As some participant nodes can be *byzantine* and behave arbitrarily with malicious intent, blockchain consensus protocols need to be byzantine fault-tolerant. In general, blockchain consensus protocols assume eventual synchrony time model [44]. The blockchain protocols typically use a broadcast channel where honest nodes receive the same set of messages with the same order. As pointed out by Cachin *et al.* [44], the blockchain form of consensus is similar to *atomic broadcast* or *total order broadcast* in crash tolerant distributed computing. Note that the blockchain consensus is not only agreeing on the total order and it involves a validation step for BFT consensus. In blockchain, consensus protocols require to follow *safety* and *liveness* properties [44, 48, 49].

- **Safety:** The safety property (or *consistency* or *common prefix* [49]) ensures that if a honest node accepts or rejects any transaction, then every other honest nodes will eventually decide for the same transaction.

- **Liveness:** The liveness property ensures that all honest nodes are guaranteed to decide for a value and terminate to reach a consensus. This assures that the blockchain grows at a steady rate.

3.2.3 A Simple Blockchain Model: How does it Work?

Current blockchain technologies include many functionalities of the Bitcoin network. Some common functionalities are transaction integrity of the ledger, prevention of double-spending, anonymity of user identity, etc. The general workflow in a blockchain is depicted in Figure 3.2. Suppose Alice wants to send some bitcoin to Bob. So Alice creates a bitcoin transaction transferring the funds to Bob, digitally signs it with her private key and broadcasts it to the network. Next step, the transaction has to be validated by the validating nodes (miners in case of Bitcoin) to check different requirement for a correct transaction (e.g., if Alice has sufficient funds). All validating nodes collect all received transactions in a block and run a consensus protocol (PoW for Bitcoin) for network approval. If the majority of the miners reach a consensus that the transactions in a block are valid, the block is appended to the blockchain. After the block containing Alice's transaction added to the blockchain, the transition from Alice to Bob become successful. The current blockchain technologies include different cryptographic techniques such as hashchain, Merkle tree, digital signatures, pseudonyms, consensus protocols to prevent double-spending attacks and create an immutable ledger.

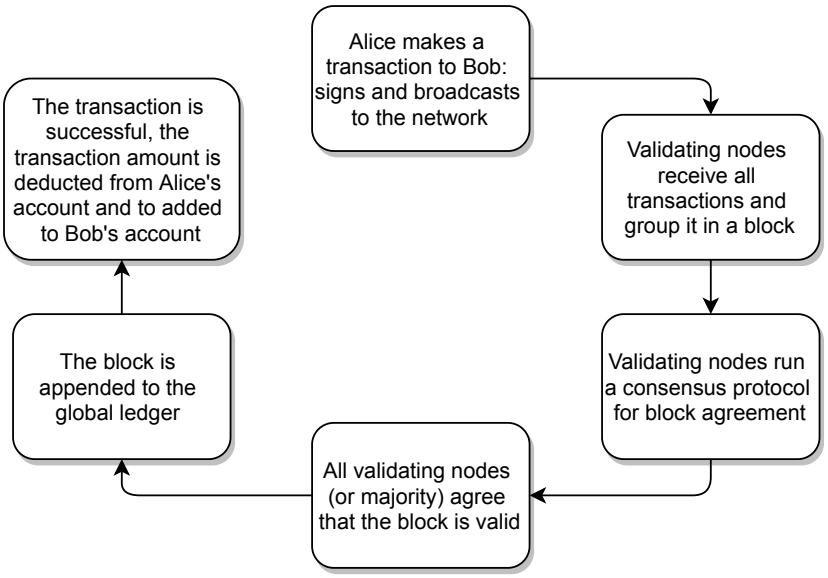

Figure 3.2: How does a Blockchain work

3.2.4 Blockchain Classification

Blockchain can be classified depending on its deployment, access rights, and verification [44]:

Permissionless Blockchain

Permissionless or *public* blockchain is open for anyone. Anyone can run a node to maintain the blockchain. Anyone can write to the shared state and add a new transaction by paying the transaction fee. Anyone can join the consensus protocol to validate correct blocks and become a miner. Some examples of permissionless blockchain are Bitcoin and Ethereum.

Permissioned Blockchain

Permissioned blockchain can be further divided into a *consortium* and a *private*. In consortium blockchain, only a pre-selected group of participants within a consortium can write to the blockchain. This restricted write permission gives that group of participants to run and influence the consensus protocol. They can also control who can issue a transaction. On the other hand, anyone can read the written transactions in the blockchain. A *private* blockchain is similar to consortium blockchain where the write permission is limited to a single participant or single organization. The read permission can be open to the public or could be limited to a subset of the blockchain users. One use case can be

data management and information sharing inside an organization. Hyperledger network [7] is an example of a permissioned blockchain.

3.3 Fork in Blockchain

A *blockchain fork* is a situation where two or more blocks have the same distance from the genesis block. Suppose, the *block height* h_b is the distance between the genesis block g and the block b such that $h_g = 0$. The *blockchain head* is a block with maximum block height $h = h_{head}$. Furthermore, let us define B_h being the set of blocks with a block height h. Then a blockchain fork happens when $|B_{head}| \geq 2$ [50]. In this situation, the nodes in the network can not reach an agreement that which block is the current blockchain head; thus, one chain becomes two or multiple chains. In other words, a fork happens when two or numerous different blocks get clear majority votes from the network participants to get accepted in the blockchain. A fork is always undesirable in any blockchain system as it could lead to a double-spending attack, create confusion in the network or reduce network performance, etc. For instance, Alice has only two bitcoins, but if the blockchain forked, she might be able to perform two bitcoin transactions to Bob and once again to Eve. In PoW based blockchains, a fork happens when two or more miners find the solution for a block around the same time. In other consensus-based blockchains, it is possible when two or more different blocks get clear majority votes. We will discuss the forks in the Ripple payment system in chapter 5. Bitcoin resolves forks in its blockchain by *longest chain rule*. It means the network will eventually select the chain with the most work and drop the others. Note that some forks can be intentionally and permanently introduced due to software upgrades or protocol changes. However, these type of forks are not relevant to our topic of discussion.

3.4 Blockchain Consensus Algorithms

We review different consensus algorithms in blockchain in focus on security and privacy.

3.4.1 Proof of Work (PoW)

Bitcoin and Ethereum employ PoW consensus each node has to perform some amount of work to add a block to the blockchain. The PoW systems use some mathematical problem, for which it is difficult to find a solution, but it is easy to verify valid solutions. To get a solution to the challenge, a *miner* node has to perform a considerable amount of computational work. The first miner finds the answer gets to add its proposed block to the blockchain and receive some reward. As verifying the correctness is easy, other nodes can agree on a correct block. The steps of PoW consensus in Bitcoin can be simplified as follows:

1. Nodes listen for the new transactions on the network, validate its correctness, and accumulate new transactions in a block.

2. Each miner node works on the PoW puzzle for its block. The puzzle includes finding a nonce such that $H(h_{prev}||h_{tx}||nonce) < target$, where H is a cryptographic hash function (SHA-256 for Bitcoin), h_{prev} is hash of the previous block, h_{tx} is the Markle root of its proposed block including new transactions and $target$ is a 256 bit number which is publicly known.

3. The first miner who solves the puzzle broadcasts its block in the network.

4. Other nodes verify the correctness of the solution, accept the block, and start working on the next block. The miner who found the correct solution receive its mining fee.

The PoW difficulty depends on the *target* value. The puzzle becomes harder when *target* value is reduced, resulting in smaller number of possible solutions. The Bitcoin network updates the *target* value in every 2016 blocks to make the puzzle more difficult. Any node can take part in PoW based consensus by starting mining; thus, it is suitable for permissionless blockchain. The miners together can make a corporate network (known as *mining pool*) to generate more hashing power thus increases the probability of finding a new block. The PoW based systems are susceptible to 51% *to attack*. This attack is possible when colluding attackers control more than 51% of the computing power in the network. PoW based consensus can support a large number of nodes; however, the transaction confirmation is slow. On average, a Bitcoin transaction takes ten minutes to get confirmed in its blockchain.

3.4.2 PBFT

Byzantine Fault Tolerant (BFT) based consensus algorithms aim to solve the consensus problem with a voting process in the presence of Byzantine nodes. Castro and Liskov showed that BFT could be practical with Practical Byzantine Fault Tolerance (PBFT) protocol in [4]. The PBFT protocol assumes that the number of Byzantine nodes $t < \frac{n}{3}$ where n is the number of total nodes in the network. The protocol is leader-based, and only the leader node (also known as *primary replica*) is responsible for committing a new block with the ordered transaction. PBFT based protocols require every node to know all other nodes participating in the consensus protocol. The primary node is selected by the other participant nodes (also known as *secondary replica*). Each round of PBFT has a *view* which is a configuration of replicas with a primary. Secondary replicas can collectively replace the primary node with a secondary node by *view-change* voting procedure if the first shows some Byzantine behavior. The PBFT protocol works in asynchronous network assuming that messages between non-faulty nodes arrive within fixed but unknown time delay. This network model is known as *eventual synchrony* and known to be a reasonable assumption for blockchain implementations [44]. The protocol can be described in three phases:

pre-prepare, *prepare* and *commit*. We can briefly describe PBFT in blockchain scenario:

1. Client sends a transaction request to the primary node.

2. In pre-prepare, the primary node assigns the transaction request with a unique sequence number and broadcasts it to secondary replicas.

3. In prepare, each non-faulty replica agree on a valid transaction (e.g., checking the signature, transaction hash) with the corresponding sequence number.

4. In the commit phase, each replica sends its commit message to other replicas for reaching consensus, executes the transaction and replies to the client.

5. Clients receives replies from the replicas and $t + 1$ identical acknowledgments confirm the transaction validation.

Some variants of PBFT algorithm is currently used in Hyperledger Fabric [7], BFT-SMaRt [51] and Tendermint [52] consensus protocol. On scalability perspective, PoW based blockchain protocols suffer from high latency for a transaction to get validated, where PBFT based protocols can support low latency in transaction validation. On the other hand, PBFT based protocols behave poorly with a higher number of nodes (currently maximum 20 nodes) thus more applicable in permissioned blockchain.

3.4.3 Consensus with Flexible Trust

In the last two sections, we discussed PoW based consensus, which is suitable for permissionless blockchain and PBFT based consensus being used in permissioned blockchain. The credit networks like Ripple [5] and its offspring Steller [53] stand in between, and their blockchains operate in a semi-permissioned manner. Both blockchains are permissionless as any node can join the consensus protocol, but each node must trust a consortium of nodes, thus somewhat similar to permissioned blockchains. This trust assumption is known as *flexible* or *subjective* or *asymmetric* trust i.e. each node must trust a group of nodes of its choice to run the consensus protocol [54].

The Ripple blockchain (known as *XRP ledger*) consensus is a voting based protocol performed by so-called *validating nodes* or *validators* in the network. Any node is open to join the network and run as a validator. Each validator requires to define a *Unique Node List* (UNL). Every validator trusts its own UNL member that they would not collude to make malicious attempts such as validate an invalid transaction. However, Ripple validators are not free to make their trust assumption as Ripple network provides "a default and recommended list" in every UNL. Thus it caused disputes over its decentralization [5]. The consensus protocol in Ripple assumes the number of byzantine nodes $t < \frac{n}{5}$

whereas traditional PBFT can tolerate upto $\frac{n}{3}$, where n is the number of consensus participants. We will discuss Ripple's consensus protocol in detail and give our security analysis in chapter 5.

The Stellar blockchain has evolved independently with similar design principle like Ripple. Similar to Ripple, it is also a credit network for cross border transactions. The *Steller Consensus Procotol* (SCP) [55] introduces *Federated Byzantine Quorum Systems* (FBQS), where any node is open to join the consensus process and can define its own trusted set of nodes known as a *Quorum Slice*. Different quorum slices may overlap and make a *Quorum* which is the sufficient number of nodes to reach consensus. For example, if a PBFT system has total nodes $n = 3t + 1$ with t byzantine nodes, the Quorum consists with $2t + 1$ nodes. Recently, Kim *et al.* [56] (Figure 2) showed current SCP deployment might fail in a sequence in the absence of two particular nodes controlled by Stellar foundation.

This idea type "UNL" or "Quorom Slice" in consensus protocols can be traced back to *Byzantine quorum systems* [57] to achieve BFT. However, Byzantine quorum systems consider symmetric trust assumption, whereas the trust assumption in Ripple or Steller is asymmetric. Recently, Cachin and Tackmann in [54] have formalized a model of Byzantine quorum systems with asymmetric trust assumption.

3.4.4 Other Alternatives

Some other alternatives have emerged in parallel for blockchain consensus mechanism. As PoW mining is costly in terms of energy usage, *proof of stake* (PoS) came as a substitution where computational power is replaced with the "stake" in the network. The idea is that the more capital a node has invested in the network, it is more likely that the node will want the network to succeed rather than attacking the system. In PoS, the probability of adding the block in the blockchain by a node is proportional to the relative stake of that node in the network. PoS can be incorporated with PoW as well as BFT based protocols.

In Algorand blockchain [58], the authors proposed a PoS Byzantine agreement protocol with *participant replacement* mechanism. Ethereum's Casper [59] is another permissionless blockchain implementation with PoS consensus.

Proof of Elapsed Time (PoET) is a consensus algorithm proposed by Intel where the computational puzzle in PoW systems get replaced by *trusted execution environment* (TEE) such as Intel's *Software Guard Extension* (SGX). The consensus mechanism is leader based, and leader must be randomly chosen to add a new block. Each node has to wait a random time interval and node with the shortest waiting time will be the leader to finalize a block.

Furthermore, SGX can verify the proof that waiting time is indeed random, and the winner has completed that time duration. Currently, Hyperledger Sawtooth platform deployed PoET in its permissionless and permissioned version. The security of PoET depends on the trusted hardware modules as an attacker nodes might perform rollback attacks and key extraction [44].

3.5 Summary

We have discussed the necessary background on blockchain, its key components, and particularly different types of consensus mechanism. We outlined how consensus in Ripple or Stellar differ from PoW and standard BFT based consensus. In the next chapters, we will talk about the Ripple credit network, its consensus algorithm, and our security analysis.

Chapter 4

Ripple Credit Network

4.1 Introduction

Ripple is a distributed payment system and global remittance service based on credit networks [11]. The initial idea behind Ripple network started with *I Owe You* (IOU). It was drafted by Ryan Fugger back in 2004 [60]. Since then Ripple payment system [61] has evolved independently of Bitcoin and gained considerable popularity over the years after its public inception in 2013. Originally, Ripple has emerged as a competitor of Bitcoin with much faster transaction validation speed (average 5 seconds). In recent times, the Ripple network has flourished more as an alternative platform for traditional cross border payment systems (e.g., SWIFT). As a cross border transaction between two banks involves hefty processing fees and considerable time, the Ripple network promises to reduce both significantly. Over 200 banks and financial institutions (e.g., CBW Bank, Royal Bank of Canada, Santander) have already adopted the Ripple network to improve its payment processing [62]. For example, Spanish bank the Santander group has estimated using Ripple could save them 20 billion US dollars annually [63].

Ripple's built-in currency *XRP* is consistently among the top three cryptocurrencies by market capital along with Bitcoin and Ethereum [64]. The XRP does not involve any mining like Bitcoin, and initially, a total of 100 billion XRP units were created to start the system. In the beginning, Ripple founders held 20% of those units, Ripple Labs retained 25%, and the remaining 55% was planned to promote the growth of the network. It is by far, one of the largest holdbacks in any cryptocurrencies; however, it did not stop Ripple to attract a large pool of users. Currently, Ripple Labs releases that 55% of total XRP for public trading into a series of escrows [65]. For instance, the XRP circulating supply is increased from 27 billion units in 2013 to 43 billion units in 2019 [64]. On January 4th, 2018, XRP market capita momentarily surpassed 148 billion US dollars and recorded each XRP unit worth 3.84 US dollars [66]. At the time of writing, Ripple network claims to have a total network value

of approximately 26 billion US dollars over 1.8 million accounts [67, 68]. This has been a significant increase in terms of network growth since the time of our study in 2015 [16] when Ripple had a market value of 960 million US dollars with little above 150,000 accounts.

One can relate Ripple's transaction mechanism to medieval *Hawala* system. At the core, Ripple's transaction model is built upon a network of pairwise trust among users in terms of credits [69]. Ripple payment system enables the transfer of traditional fiat currency (e.g., USD, Euro) together with cryptocurrencies (e.g., BTC, XRP) through IOU credit paths in user-defined currency. An IOU transaction is only possible if there exists a credit path between two users. In 2014, the International Ripple Business Association deployed several Ripple gateways and market makers [70] around the world to make Ripple credit graph complete. The *Ripple ledger* (also known as *XRP ledger*) is an immutable public blockchain which keeps track of the transactions, account information, credit links since the genesis block. The *Ripple consensus protocol* [5] supports its blockchain and helps participants to agree on a new block containing a set of transactions. However, Ripple's consensus protocol is non-standard and differs from traditional PBFT consensus [4], which makes its security an open problem.

The presented work in this chapter and the next chapter is based on the January 2015 version of Ripple protocol and previously published description in [16]. In this chapter, we give an overview of the Ripple system and related works on its security and privacy.

4.2 Overview of Ripple

In the following, we introduce and give details of the Ripple credit network and its blockchain components.

4.2.1 Key Roles in Ripple Network

The Ripple code is available for the public and open-source so that anyone can deploy a Ripple instance. Ripple nodes can take a few different roles in the system as follows:

User

A user in the network which runs the Ripple client application to make/receive payments. A user has a public-private key pair for signing transactions. Ripple users are referenced with its public key address as pseudonyms. If a user wants to make a payment to another user, she can cryptographically sign the transfer of money in XRP or other currency. Ripple has no way to enforce non-XRP payments, and only records the amounts owed by one entity to the other.

Validating server

The *validating servers* (also knowns as *validators*) execute Ripple consensus protocol to check and validate all transactions in the system. Other than validation, nodes running *rippled server* software can receive and relay messages and running back-end applications, etc. Each validating server has a public-private key pair for transaction validation.

Gateway

A *gateway* is a reputed user who helps to create trust links when a new user joins the network. For instance, a new user creates a new account wallet with her public-private key pair upon registering in the network. However, it is not enough to use the system as she needs to have some trust links with other users and some account balances (known as *bootstrapping problem*). Ripple addresses this problem by deploying several gateways in the network. These gateways are highly connected and aim to make Ripple credit graph complete. Therefore a new user can trust a gateway to make a credit link and start sending IOU transactions to other parties.

Market Maker

The *market makers* act as the trade enablers for cross-currency transactions in the system. They are equivalent to currency exchange services in the physical world. For instance, a market maker receives payment in some specific currency through one of its credit links. Then she can initiate an exchange in different currency through another credit link.

4.2.2 Credit Network

A credit network [69, 71] can be presented with two weighted directed graphs $G(V, E_1)$ and $G(V, E_2)$ sharing same set of vertices V. The set of vertices V are the user nodes in the network. The set of edges E_1 presents pairwise account balances between users. More specifically, a edge weight $o_{i,j} \in E_1$ denotes the IOU obligations that the user i has to user j. In the other graph, the set of edges E_2 shows pairwise credit limit between users. An edge $c_{j,i} \in E_2$ represents the credit line (or trust line) which quantifies the amount of credit user j has trusted to give to user i. Hence, an direct IOU payment worth of $o_{i,j}$ units from user i to j is possible if $o_{i,j} \leq c_{j,i}$. Note that i and j do not need to be neighbors to send IOU payments as long there exists a path with sufficient credit limit.

4.2.3 How does Ripple Work?

The basic idea of Ripple is that of a public billboard containing IOUs. Whenever a sender Alice (or A) wants to issue a payment to a receiver Bob (or B), she puts up a signed message with the relevant transaction data to the billboard. This way, the billboard keeps track of all the transactions in the community.

In the case of Ripple, the "billboard" is a distributed database known as *ledger* or *blockchain* that is managed by a peer to peer (P2P)network. Thus, a new IOU is not sent to the receiver directly, but to the Ripple P2P network where each peer keeps its own copy of the ledger. If a clear majority of the servers agrees that the IOU indeed constitutes a valid payment, then it becomes a valid part of the ledger. Since Ripple itself has no power to enforce actual payments (it only keeps track of payment *promises*), the system relies on transitive trust from one user to another.

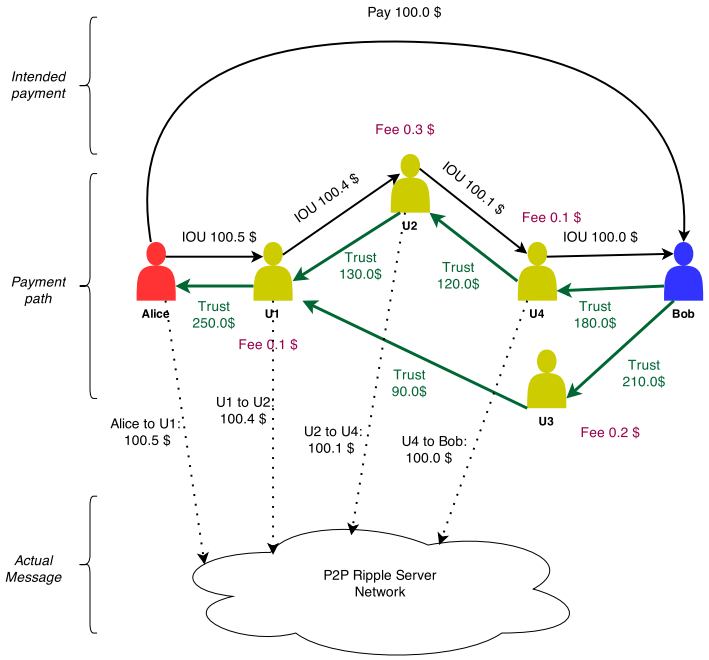

Figure 4.1: (Intended Payment) Alice wants to pay 100.0 $ to Bob. (Payment Path) Since Bob does not accept IOUs directly from Alice, she has to get help from intermediaries, who in turn may charge a fee. (Actual Messages) The real IOU messages are not sent between the involved parties directly, but to the server network.

An IOU payment from A to B is only possible if B is willing to accept an IOU from A, i.e., B trusts A and give enough credit to A. Therefore, A can only make a correct payment to B if the payment value is less than equal to the credit balance allocated by B to A. To make such type of transaction model work, both users need to know each other beforehand, or the amount should be small. However, it requires some intermediary such as a market maker to process a typical payment in the Ripple system. Therefore, Ripple credit graph must have good *connectivity* and *liquidity* to perform payments between arbitrary users [71].

Figure 4.1 illustrates how a typical IOU payment in Ripple credit network looks. The actual payment is procured by sending a sequence of payments along a payment path, where intermediaries may receive a fee for their service. The requirement to make a payment successful is that enough credit must be available throughout the payment path. To start using the system, A can deposit some amount to a market $U1$ to generate a trust line between $U1$ to A. For instance, Alice (A) intends to make a payment of 100 US dollars to Bob (B). As A and B are not neighbors, the payment is routed through the path $A \rightarrow U1 \rightarrow U2 \rightarrow U4 \rightarrow B$. This payment is possible as each credit line in the path is larger than the payment amount for each atomic transactions. Note that we could not route through $U3$ because of insufficient credit limit given to $U1$ from $U3$. Nevertheless, it is possible to distribute the amount such that 90 US dollars through $U1 \rightarrow U3 \rightarrow B$ and the rest through $U1 \rightarrow U2 \rightarrow U4 \rightarrow B$. In this case, A needs to spend more as $U3$ requires an additional fee. As market makers have different fees, Ripple employs a path-finding algorithm for the most suitable path from source to destination.

Also, note that there is no payment message directly from A to B. Instead, the atomic unit of a Ripple transaction is the single payment message sent from the users to the network. Additionally, Ripple's currency XRP can be used as a bridge currency in case of currency pairs that are traded rarely.

4.2.4 Ripple Ledger

Ripple ledger or XRP Ledger is the public blockchain of the Ripple network. This distributed ledger keeps track of all the transactions made in the system. A new ledger (or a block) is created in every few seconds containing a set of transactions agreed by the majority of *validating servers*. The agreement process is achieved through Ripple consensus protocol [5] which is executed among validating servers. The information stored in a Ripple ledger L can be categorized as follows: *(i)* a set of transactions, *(ii)* account-related information such as account settings, total balance, trust lines, *(ii)* a timestamp, *(iv)* a ledger number, and *(v)* a status bit which indicates whether the ledger is validated or not. A validated ledger with most recent timestamp is known as *last closed ledger*. If the status bit is 0, we call it *open ledger*, which reflects the current operational ledger and not it yet confirmed by the network.

4.2.5 Ripple Transactions

At the time of writing [16], Ripple supported six types of transactions [72] as follows:

Payment: This is the most regular type of transactions that allows a user to send funds from one account to another.

AccountSet: This transaction grants a user a set options relevant for one's account. Note that it is possible to cancel a transaction with the same

`SequenceNumber` by an `AccountSet` transaction given that the transaction has not been validated in a ledger yet.

SetRegularKey: This transaction permits a user to change/set its previously used key to sign future transactions.

OfferCreate: This transaction initiates an intent to exchange currencies.

OfferCancel: This transaction withdraws an offer from the ledger.

TrustSet: This transaction allows to change/set a trust link between two user accounts.

As presented in Table 4.1, all six types of transactions contain some standard fields. When a new user opens an account in Ripple, she would require to issue a payment of minimum 20 XRP for account activation. After this transaction is validated in the global ledger, that new account will be included in the global ledger.

4.2.6 Ripple Consensus Process

Ripple network includes new blocks in its blockchain known *Ripple Ledger* by a round-based consensus mechanism. The Ripple consensus protocol is carried out by a set of validating servers. After running the consensus protocol, they decide on which set of transactions to add to the Ripple ledger. First, a transaction operation is initiated by a user when she signs it with her private key. The transaction is then forwarded to validating servers and seeks validation. The validating servers, on the other hand, are responsible for checking the correctness of the transactions received from different users. The correctness of a transaction depends on the current state of the ledger and consensus rules. Due to the asynchronous nature of the network, validating servers may receive a different set of transactions from each other. Finally, the goal of the consensus is to reach an agreement on the set of transactions.

In a round-based process, each validating server first broadcasts its own set of proposed transactions known as *candidate set* to other peers. Among many unconfirmed candidate sets, the final validation comes through a round-based voting process. In the first iteration, the voting requirement is 50% of the validating servers and later on increased to 60%, 70%, and 80% in each round. When a candidate set receives 80% votes, validating servers add the candidate set in the global ledger. Note that any transaction that has been initiated during the consensus rounds but did not appear in the ledger is discarded and considered as invalid by the users.

Each validating server maintains a list of servers known as *Unique node list* (*UNL*) including herself that she *collectively trust* or *subjectively trust*. More specifically, the *collective trust* symbolizes that a server trusts a set of servers who will not "collude to defraud" [5]. Therefore, servers only trust the votes issued by other servers in her *UNL*.

4.3 Related Works

To the best of our knowledge, our work in 2015 [16] was the first analysis of the Ripple payment system. In [16], We analyzed the Ripple ledger consensus protocol and found that the parameters chosen by Ripple designers did not prevent a fork in the network. We will discuss our results on Ripple ledger consensus protocol from [16] in the next chapter. Furthermore, an analysis of usage patterns and trade dynamics of Ripple ledger till 2015 is given in [16]. The first observation was that the number of transactions in the Ripple platform had increased from late 2014. However, most transactions ($> 70\%$) relate to OfferCreate and OfferCancel rather than an actual XRP transaction. It showed that Ripple was used more as a platform for currency exchange instead of actual payments. Second, the majority of the accounts contain a small amount of XRP and barely performed transactions. Third, most ledger indeed closes within a few seconds where a few could take up to 30-40 sec maximum. Interested readers are recommended to read [16] for in-detailed results in usage patterns and trade dynamics of the Ripple network.

There exist some works to improve privacy in Ripple like credit networks in [71, 73, 74, 75, 76, 77, 78]. In [71], Moreno-Sanchez *et al.* presented a provably secure privacy-preserving protocol for credit networks. Their work introduces the different privacy notions (e.g., value privacy, sender/receiver privacy) for credit networks. Furthermore, the authors in [71] enforce privacy in transactions in a centralized manner with trusted hardware and oblivious RAM. Malavolta *et al.* in [73] proposed a distributed privacy-preserving protocol for credit networks called *SlientWhispers* which uses SMC in the universal composability (UC) framework [30, 79].

On Ripple transaction anonymity, authors in [78] proposed two types of heuristics based on the Ripple network graph to perform deanonymization attacks. The first heuristic could link Ripple accounts with Bitcoin addresses, whereas the second one identifies co-related Ripple accounts by the same user.

Luzio *et al.* in [75] have analyzed the statistics of the first three years of Ripple network usage (till September 2015). They studied the consensus agreement data and behavior of Ripple network participants and concluded most of the blocks indeed are validated by a small number of validating servers hosted by Ripple Labs. They also showed a new deanonymization approach to link users identities from Ripple transaction fingerprints. In PathSuffle paper [80], authors proposed a mixing protocol for the Ripple network to anonymize path-based transactions. In order to improve privacy and routing in credit settlement in Ripple, Roos *et al.* in [74] designed a privacy-preserving routing algorithm *SpeedyMurmurs* which achieves efficiency improvement by a factor of two for path-based transactions in Ripple network.

Moreno-Sanchez *et al.* in [77] analyzed statistics of Ripple network up to August 2017. It showed that the number of Ripple accounts and credit link have grown substantially in 2017 and accessed the health of the network. They estimated 13M USD are at risk due to the inappropriate setting of the flag known as *no-ripple* on the credit links. Furthermore, they observed more than

112,000 accounts are prone to isolation from main Ripple network, and its credit are at risk due to insufficient gateway connections.

The Ripple consensus protocol [5] assumes *Byzantine accountability*, which enforces a limitation on byzantine nodes such that sending different messages to different nodes is not permitted. Recently in 2018, Chase and MacBrough [81] suggested this assumption is not practical due to network partitioning and asynchrony. We will discuss the results of [81] in the next chapter.

4.4 Summary

We gave an overview of Ripple credit network and its blockchain specifications. Additionally, we present the related works in the security and privacy aspects of Ripple. We detail and analyze Ripple consensus protocol in the next chapter.

Field	Internal Type	Description
Account	Account	The unique address of the account that initiated the transaction.
AccountTxnID	Hash256	(Optional) Hash value identifying another transaction. This field allows the chaining of two transactions together so that a current transaction is only valid unless the previous one (by Sequence Number) is also valid and matches the hash.
Fee	Amount	(Required) Integer amount of XRP, in drops, to be destroyed as a fee for distributing this transaction to the network.
Flags	UInt32	(Optional) Set of bit-flags for this transaction.
LastLedgerSeq	UInt32	(Optional) Highest ledger sequence number that a transaction can appear in.
Memos	Array	(Optional) Additional information used to identify this transaction.
Sequence	UInt32	(Required) A transaction is only valid if the sequence number is exactly 1 greater than the last-validated transaction from the same account.
SigningPubKey	PubKey	(Required) ASCII representation of the public key that corresponds to the private key used to sign this transaction.
SourceTag	UInt32	(Optional) Arbitrary integer used to identify the reason for this payment.
TransactionType	UInt16	The type of transaction.
TxnSignature	VariableLength	(Required) Transaction signature.

Table 4.1: Standard fields present in all Ripple transaction types [16]

Chapter 5

Analysis of Ripple's Consensus Protocol

5.1 Introduction

In this chapter, we describe Ripple's consensus protocol and give our analysis on its security. Primarily, our contribution is on the safety property of Ripple's consensus protocol, which is previously published in [16]. The following Ripple protocol description is based on Ripple client (version 0.2.48-2), *Rippled* server (version 0.23.0), original Ripple white paper [5], the Ripple Wiki [82], and our publication in [16]. After presenting our analysis on safety and fork in Ripple blockchain, we will also discuss some improvement of our results by Chase and MacBrough [81].

5.2 Ripple Workflow

The Ripple client requires an email id to create the Ripple wallet, and the user gets a public-private key pair. The account only gets activated, i.e., network stores its public account address (derived from public key) in the blockchain, when it receives more than some fixed amount of funds from another existing Ripple account. The ripple network consists of servers that run a P2P server software known as *Rippled*. If a user wants to make some payment to someone, he creates a transaction and signs it with his secret key. Then, the client submits the transaction to a trusted (locally operated) Ripple server so the user can be sure the transaction is submitted in the network. Now, this provisional transaction will be forwarded by different servers throughout the network in a P2P fashion. Upon receiving the transaction, servers will check if it is well-formed, and if the transaction is malformed, it will be rejected instantly. If the transaction is well-formed, servers will run a consensus protocol to include it to the ledger permanently.

5.2.1 Protocol Components

Any entity that runs the Ripple client application is a **participant** in the Ripple network. The participant could be a gateway, web wallet, or financial bank, etc. A participant has a public-private key pair for signing transactions.

Any **server** in the network runs the Rippled server software. Servers are used for running consensus protocol, receive and relay messages, running back-end applications, etc. Our main focus is on **validating servers**, which take part in consensus protocol in addition to performing other server roles. It has a separate public-private key pair for validation. We use the notation v to refer to a validating server. Note that the validating servers might be referred to as just servers or nodes in this chapter description.

The network has a shared ledger which records all information about transactions: the states of account balances, account details, different offers, timestamp, etc. The ledger gets continuously updated and stores all transactions, which passes the consensus phase. A five-element tuple can represent a Ripple **ledger** L:

$$L = (\tau, \lambda, \gamma, \eta, \alpha),$$

where τ is a set of transactions; λ denotes set of different information like account settings, balances, trust information; γ is the timestamp; η is the ledger number, and $\alpha \in \{0, 1\}$ signifies if the ledger L is validated or not. If α is 1, we call it a validated ledger (published). A validated ledger with the most recent timestamp is known as **last closed ledger**. If α is 0, we call it **open ledger**, which reflects the current operational ledger and not it yet confirmed by the network.

Each validating server maintains a set of trusted servers known as the **unique node list** (*UNL*), including itself. Note that it not necessary that every server is connected with another server in the network. We denote, *UNL* of server v as UNL_v. Only the votes from the members of UNL_v is relevant when v checks for consensus of a transaction. During the consensus protocol, any v can broadcast a **proposal** i.e., a set of transactions that v proposes to be included in the public ledger L. Another validating server u considers a proposal from v only if $v \in UNL_u$.

5.2.2 Sending a Payment in the Network

Similar to Bitcoin, Ripple uses 256 bit ECDSA keys and the Secp256k1 elliptic curve for its digital signature scheme. In the following, by H and $Sign$, we denote a hash function and a signature algorithm, respectively. Let X be an arbitrary user in the network. Upon joining the Ripple network, X generates Ripple wallet with a public key pk_X and a corresponding secret key sk_X. The public key is hashed to form the pseudonymous identity of X, namely ID_X $(= H(pk_X))$ and stored in the distributed ledger L (in λ). Consider a sender A and a receiver B. Then an IOU is issued as follows:

1. A composes a transaction message:

$$t = (ID_A, ID_B, trans\_data, seq\_nr, pk_A),$$

where *trans_data* includes transaction data like payment amount and relevant fees, and *seq_nr* is a A's current IOU sequence number.

2. A signs hash of the message:

$$\sigma_t = Sign(sk_A, H(t)).$$

3. A submits the signed message (t, σ_t) to the network and it gets forwarded by the servers.

5.3 The Ripple Consensus Protocol

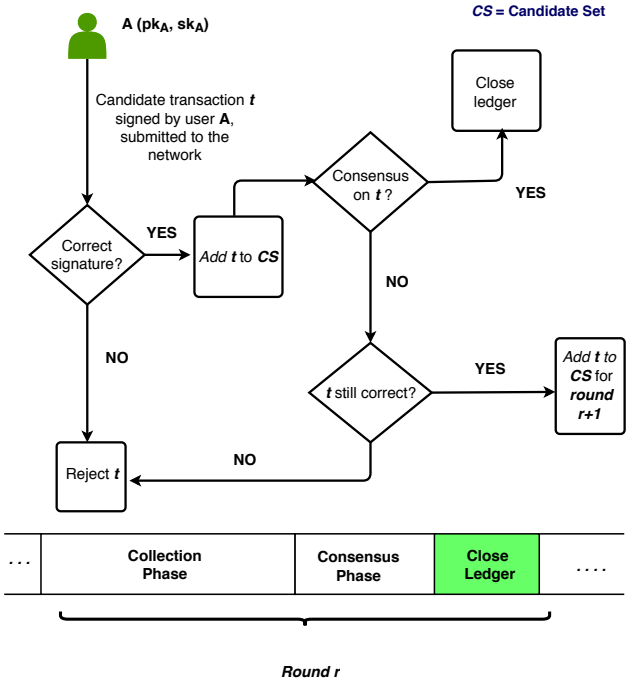

Figure 5.1: Phases of round-based consensus protocol

5.3.1 Network Assumptions

The Ripple consensus protocol assumes *byzantine accountability* as mentioned in the Ripple consensus white paper [5]. This accountability assumption is a

limitation on byzantine nodes such that sending different messages to different nodes is not permitted. Also, the trust between node u and its unique node list UNL_v is asymmetric. For instance, it's possible that a node $u \in UNL_v$ but $v \notin UNL_u$.

5.3.2 Protocol Overview

When transactions are submitted to the network, they are forwarded peer to peer by different servers in the network (receive and relay servers). The goal is to update the ledger with all new payment information periodically, and usually, this happens less than 20 seconds in most cases. Since the ledger is a distributed database, the participants should reach an agreement on the transactions to include it in the next version of the ledger. This consensus protocol is run by the network's validating servers and is the core building block of Ripple. The protocol is asynchronous and round-based. At the end of every round, a new last closed ledger is published by all participating servers. The Ripple's consensus protocol has three phases (Figure 5.1): the *collection phase*, the *consensus phase*, and the *ledger closing phase*. If any consensus round takes more than a fixed amount of time, it gets aborted, and a new consensus round starts. Furthermore, the *consensus phase* is divided into sub-rounds. The threshold for voting on a transaction is increased in every sub-round. As the protocol is asynchronous, time synchronization helps to achieve better ledger closing time.

Collection Phase

In the collection phase, the validating servers accumulate the transactions that they receive from the network. Remember that the candidate transactions are broadcasted in the network. Upon receiving a candidate transaction, the validating servers verify its authenticity. To do that, they verify the issuer's public key (from the ledger), and they check the validity of the corresponding signature. Transactions that come with valid signatures are temporarily stored in the *candidate set* CS for further validation.

The validating servers then subsequently inspect the correctness of transactions stored in CS. This procedure includes verifying that sufficient credit is available in the issuing account by going over the history of all transactions related to that account (in case of XRP transactions), or availability of a trust path between the sender and receiver (for IOU payment), etc. For simplification, we denote the transaction verification actions by a single function $Verify(transaction, ledger)$. Then each validating server packages validated transactions (voted by more than a certain threshold θ percent of its UNL) in an (authenticated) proposal, and broadcasts its proposal in the network. In Ripple, this is performed by constructing a hash tree of all validated transactions, and then signing the root of the tree.

More specifically, when a validating server v receives a new proposal from the network, it checks if the issuer is a server which belongs to its UNL, and does correctness verification of all transactions included in the received proposal. For

Input: Ledger L, signed transaction (t, σ_t), candidate set CS, transaction list TL_v, vote list Vote_t, vote threshold θ, unique node list UNL_v, and v's private key sk_v

Output: Update CS, TL_v, Vote_t, create proposal (P, σ_P), and broadcast (P, σ_P)

1 $P \leftarrow \emptyset$
2 v receives candidate transaction (t, σ_t) with a valid signature
3 $CS \leftarrow CS \cup \{t\}$
/* Other checks (trust path, credit balance, etc.) on t from the ledger history */
4 **if** $\mathit{Verify}(t, L) = \text{true}$ **then**
5 $\quad TL_v \leftarrow TL_v \cup \{t\}$
6 $\quad \text{Vote}_t \leftarrow \text{Vote}_t \cup \{v\}$
7 **foreach** $t \in TL_v$ **do**
8 $\quad$ **if** $\left(\frac{|\text{Vote}_t|}{|UNL_v|} \geq \frac{\theta}{100}\right)$ **then**
9 $\quad\quad P \leftarrow P \cup \{t\}$
10 **end**
11 $\sigma_P \leftarrow \mathit{Sign}(sk_v, H(P))$
12 v broadcasts proposal (P, σ_P) to other servers in the network

Algorithm 1: Processing a Transaction and Creating a New Proposal

the positive case, these transactions are added into a locally managed *transactions list* TL_v. Furthermore, the server v maintains a vote list Vote_t for every transaction $t \in TL_v$. The server v also updates the vote list Vote_t according to the received proposal (see Algorithm 2). For instance, if the transaction t is part of the proposal P received from a server w ($t \in TL_v$ and $w \in UNL_v$), v will register a vote from w in the vote list Vote_t.

As outlined in Algorithm 1, an incoming candidate transaction t is processed for verification and added to the candidate set. After that, each validating server creates a proposal based on its *UNL* vote and broadcast to the network. The Algorithm 2 shows the processing of a new proposal, update mechanisms of the transaction list, and vote list. CS and TL_v are initially empty at round zero (genesis ledger) but continually updated in each round.

Consensus Phase

During the consensus phase, a validating server frequently processes (Algorithm 2) and sends proposals (Algorithm 1) to the network. First, v sends proposals which are agreed by more than θ percent of the servers in its *UNL*. Initially, the value of the voting threshold θ is fixed at 50% of the servers in the *UNL*. Then θ is gradually increased in each iteration by 10% until a proposal reaches consensus from 80% . The iterations of consensus phase sub-rounds are triggered by a local timer maintained by each validating server.

Input: Ledger L, proposal (P, σ_P) from a server w, transaction list TL_v, vote list Vote_t, and unique node list UNL_v
Output: Updated TL_v and Vote_t

```
1  v receives an authenticated proposal (P, σ_P) from w
2  if w ∈ UNL_v then
3  |  foreach t ∈ P do
4  |  |  if Verify(t, L) = true then
5  |  |  |  TL_v ← TL_v ∪ t
6  |  end
7  foreach t ∈ TL_v do
8  |  if t ∈ P then
9  |  |  Vote_t ← Vote_t ∪ {w}
10 |  else
11 |  |  Vote_t ← Vote_t \ {w}
12 end
```

Algorithm 2: Processing a New Proposal

Input: Last closed ledger L, candidate set CS, transaction list TL_v, vote list Vote_t, unique node list UNL_v, and v's private key sk_v
Output: Current closed ledger L, updated CS, TL_v, and Vote_t

```
1  L_v ← L
2  foreach t ∈ TL_v do
3  |  if (|Vote_t| / |UNL_v| ≥ 0.8) then
4  |  |  CS ← CS \ {t}
5  |  |  if t ∉ L_v then
6  |  |  |  L_v.apply(t)
7  |  |  |  TL_v ← TL_v \ {t}
8  |  |  |  Vote_t ← ∅
9  end
10 σ_{L_v} ← Sign(sk_v, H(L_v))
11 v broadcasts its (L_v, σ_{L_v})
12 foreach u ∈ UNL_v do
13 |  v receives (L_u, σ_{L_u})
14 end
15 Find the ledger L' among L_u's with valid signature which has clear
     majority (more than 80%)
16 L ← L'
```

Algorithm 3: Close the Ledger

Ledger Closing Phase

As shown in Algorithm 3, once a transaction t reaches 80% acceptance rate, t gets removed from the candidate set CS, checked for double-spending (i.e., examined against the transactions included in the ledger). Next, this transaction will be appended to the open ledger (L.apply(t)), and the account balance of the sender/recipient will be accordingly updated. Every validating server v will broadcast a signed hash of its version of the ledger to the network. A ledger is treated as validated (and closed) by a server v when a clear majority 80% of validating servers which are contained in v's *UNL* also sign the same ledger L. After closing the ledger, the transactions that have been received during the consensus phase will be processed, and the next round will begin.

5.4 Analysis of Forks in Ripple

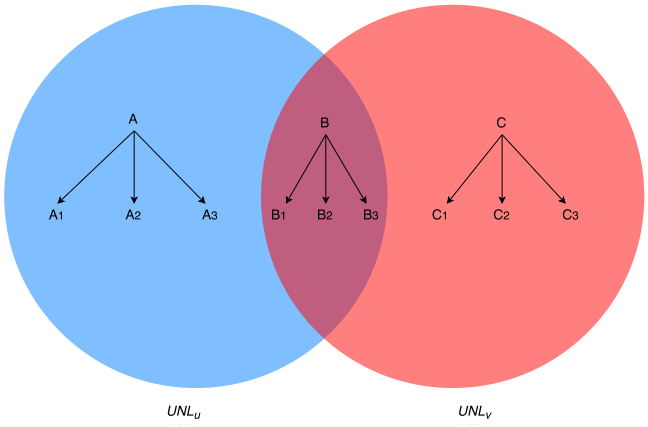

Figure 5.2: UNL intersection between two servers u and v

The security of Ripple relies on the fact that the majority of the validating servers are honest and correctly verify all the received transactions. Ledger forks constitute a major threat to the correct operation of the system. Forks can occur if two conflicting ledgers get clear majority votes, and could lead to double-spending attacks [83].

Ripple designers claim that fork cannot happen if intersection size of *UNL* of any two servers is bigger than 20% of their individual *UNL* size [5]. For any two validating server u and v:

$$|UNL_u \cap UNL_v| \geq \frac{1}{5} \max\{|UNL_u|, |UNL_v|\} \tag{5.1}$$

However, recently several forks [84, 53] lead to serious concerns about the correctness of the Ripple consensus protocol and the requirements for no fork

guarantee in the system. In this section, we take a deep dive into the conditions which can prevent forks in Ripple network. For any two validating servers u and v, suppose:

$$w_{u,v} = \frac{|UNL_u \cap UNL_v|}{\max\{|UNL_u|, |UNL_v|\}} \tag{5.2}$$

Current Ripple protocol mandates for any two validating servers u and v, the value of $w_{u,v}$ must be greater than or equal to 0.2. Clearly, We now show that this threshold is not sufficient to prevent forks in the system by means of a counter-example. Namely, consider the situation where for some u and v, $|UNL_u| = |UNL_v| = 5$ and $|UNL_u \cap UNL_v| = 2$. Clearly, $w_{u,v} = 0.4 \geq 0.2$. Assume now that one server in $UNL_u \cap UNL_v$ votes for L_1 and the other for (conflicting ledger) L_2. Moreover, assume that all servers in $UNL_u \setminus UNL_v$ vote for L_1 and similarly all servers in $UNL_v \setminus UNL_u$ vote for L_2. This means that a majority of 80% in UNL_u vote for L_1 and likewise a majority of 80% in UNL_v vote for L_2. This clearly results in a fork in the system.

As this example shows, the condition displayed in Equation 5.2 cannot prevent forks in general for values $w_{u,v} \leq 0.4$. In the following, we will prove that if the intersection set size between the *UNL* of any two servers is more than 40% of size of the largest *UNL*, that is $w_{u,v} > 0.4$, then forks in Ripple are impossible. In other words, forks in Ripple are impossible *if and only if* for all validating servers u and v:

$$|UNL_u \cap UNL_v| > 0.4 \cdot \max\{|UNL_u|, |UNL_v|\} \tag{5.3}$$

We denote the threshold value for any transaction to get clear majority votes by ρ where $0.5 < \rho \leq 1$.

Recall that a fork refers to the situation that two different validating servers u and v agree on conflicting ledgers $L_1 \neq L_2$. This means that at least a fraction ρ of servers in UNL_u agree on ledger L_1 and at least a fraction ρ of servers in UNL_v agree on ledger L_2. We consider the following sets:

$$A := UNL_u \setminus UNL_v, \quad B := UNL_u \cap UNL_v, \quad C := UNL_v \setminus UNL_u. \tag{5.4}$$

For each server $\in UNL_u \cup UNL_v$, three possible cases might occur:

Case 1: The server publishes ledger L_1.

Case 2: The server publishes ledger L_2.

Case 3: The server does not publish or publishes some ledger other than L_1 and L_2.

In the following, we denote the subset of servers in set A that publish L_1 by A_1, the subset of servers in A that publish L_2 by A_2, and the subset of servers that publish neither L_1 nor L_2 by A_3. Clearly, A_1, A_2, and A_3 are mutually exclusive, and $|A_1| + |A_2| + |A_3| = |A|$. Analogously, we define the sets B_1, B_2, B_3, C_1, C_2, and C_3 (cf. Equation 5.4).

Necessary Conditions for Forking: According to the specification of Ripple, it holds that if more than a fraction ρ of the servers present in any server's *UNL* publishes the same validation ledger hash, that ledger will be accepted by that server. Hence,

1. Ledger L_1 will be accepted by server u if and only if

$$\begin{aligned} |A_1|+|B_1| &\geq \rho(|A|+|B|) \\ |A_1|+|B_1| &\geq \rho(|A_1|+|A_2|+|A_3|+|B_1|+|B_2|+|B_3|) \\ \Leftrightarrow (1-\rho)(|A_1|+|B_1|) &\geq \rho(|A_2|+|A_3|+|B_2|+|B_3|) \\ \Leftrightarrow |A_1|+|B_1| &\geq \frac{\rho}{1-\rho}(|A_2|+|A_3|+|B_2|+|B_3|) \end{aligned} \tag{5.5}$$

2. Likewise, ledger L_2 will be accepted by server v if and only if

$$|B_2|+|C_2| \geq \frac{\rho}{1-\rho}(|B_1|+|B_3|+|C_1|+|C_3|) \tag{5.6}$$

Minimum Intersection Size: Notice that a fork is only possible if both Equations 5.5 and 5.6 are satisfied. Recall $w_{u,v} = \frac{|UNL_u \cap UNL_v|}{\max\{|UNL_u|,|UNL_v|\}}$, we show that $w_{u,v} \geq 0.4$ ensures fork cannot occur in Ripple.

Observe that:

$$\begin{aligned} |UNL_u \cap UNL_v| &> w_{u,v} \cdot |UNL_u| \\ |B_1|+|B_2|+|B_3| &> w_{u,v}(|A_1|+|A_2|+|A_3|+|B_1|+|B_2|+|B_3|) \\ (1-w_{u,v})(|B_1|+|B_2|+|B_3|) &> w_{u,v}(|A_1|+|A_2|+|A_3|) \\ (|B_1|+|B_2|+|B_3|) &> \frac{w_{u,v}}{1-w_{u,v}}(|A_1|+|A_2|+|A_3|) \end{aligned} \tag{5.7}$$

Similarly, we have:

$$(|B_1|+|B_2|+|B_3|) > \frac{w_{u,v}}{1-w_{u,v}}(|C_1|+|C_2|+|C_3|) \tag{5.8}$$

Now, adding Equations (5.7) and (5.8) we get,

$$(|B_1|+|B_2|+|B_3|) > \frac{w_{u,v}}{2(1-w_{u,v})}(|A_1|+|A_2|+|A_3|+|C_1|+|C_2|+|C_3|) \tag{5.9}$$

Assuming that both Equations 5.5 and 5.6 are satisfied, it follows that:

$$\begin{aligned} |A_1|+|B_1|+|B_2|+|C_2| &\geq \frac{\rho}{1-\rho}(|A_2|+|B_2|+|B_1|+|C_1|+|A_3|+|C_3|) \\ &\quad + \frac{2\rho}{1-\rho}|B_3| \\ |A_1|+|C_2| &\geq \frac{\rho}{1-\rho}(|A_2|+|C_1|+|A_3|+|C_3|) \\ &\quad + \frac{2\rho-1}{1-\rho}(|B_1|+|B_2|+|B_3|) + \frac{1}{1-\rho}|B_3| \end{aligned} \tag{5.10}$$

Combining Equations 5.9 and 5.10, we get the following strict inequality:

$$\begin{aligned} |A_1| + |C_2| \quad > \quad & \frac{\rho}{1-\rho}(|A_2| + |C_1| + |A_3| + |C_3|) \\ & + \frac{(2\rho-1)w_{u,v}}{2(1-\rho)(1-w_{u,v})}(|A_1| + |A_2| + |A_3| + |C_1| + |C_2| + |C_3|) \\ & + \frac{1}{1-\rho}|B_3| \end{aligned}$$

This can be rephrased to:

$$(1 - \frac{(2\rho-1)w_{u,v}}{2(1-\rho)(1-w_{u,v})}) >$$

$$\underbrace{\frac{1}{(|A_1| + |C_2|)}}_{\geq 0} \cdot \left[(\underbrace{\frac{\rho}{1-\rho}}_{\geq 0} + \underbrace{\frac{(2\rho-1)w_{u,v}}{2(1-\rho)(1-w_{u,v})}}_{\geq 0}) \underbrace{(|A_2| + |A_3| + |C_1| + |C_3|)}_{\geq 0} + \underbrace{\frac{1}{1-\rho}|B_3|}_{\geq 0} \right]$$

As already marked, the right-hand side is ≥ 0. Hence, this cannot hold if:

$$\begin{aligned} (1 - \frac{(2\rho-1)w_{u,v}}{2(1-\rho)(1-w_{u,v})}) \quad & \leq \quad 0 \\ (2-2\rho)(1-w_{u,v}) - (2\rho-1)w_{u,v} \quad & \leq \quad 0 \\ (2-2\rho-w_{u,v}) \quad & \leq \quad 0 \\ w_{u,v} \quad & \geq \quad 2(1-\rho) \end{aligned}$$

As a consequence, if $|UNL_u \cap UNL_v| > 2(1-\rho)\max\{|UNL_u|, |UNL_v|\}$ for all pairs of validating servers u, v, then no fork can occur in Ripple for sure. Since $\rho = 0.8$ in the current Ripple system, a sufficient condition for preventing forks is to ensure $w_{u,v} > 0.4$ for all validating servers u and v.

5.5 Analysis from Chase and MacBrough [81]

As aftermath, Ripple Labs have agreed and acknowledged our security analysis in their public web-page [85] in 2015. Following our work [16], Chase and MacBrough have extended the analysis of Ripple's consensus protocol in 2018 [81]. In the last section, the necessary condition for no fork was established with a counter example. We showed that if the intersection does not have more than 40% of the maximum size between two UNLs, a fork might happen. We saw that a fork can happen when $|UNL_u| = |UNL_v| = 5$ and $|UNL_u \cap UNL_v| = 2$ with an overlap exactly 40% of the maximum size between two UNLs. This example implicitly assumes that both UNLs have same size. Chase and MacBrough have shown that the necessary condition can be relaxed by taking average instead of maximum [81]. The necessary and sufficient condition can be restated as:

$$|UNL_u \cap UNL_v| > 2(1-\rho) \cdot \ \text{avg}\{|UNL_u|, |UNL_v|\} \tag{5.11}$$

For simplicity let's denote, $|UNL_u| = n_u$ and $|UNL_v| = n_v$. The condition can be rewritten as: $|UNL_u \cap UNL_v| > (1-\rho)(n_u + n_v)$.

To show this condition is necessary, we can have two cases:

Case 1: $|B| = |UNL_u \cap UNL_v| \leq (1-\rho)n_v$. In this scenario, it is possible for all servers in UNL_u to publish ledger L_1 and on other hand all servers in $C = UNL_v \setminus UNL_u$ can publish another ledger L_2 as $|C| \geq \rho \cdot n_v$, which results in a fork.

Case 2: $|UNL_u \cap UNL_v| \leq (1-\rho)(n_u+n_v)$ and $|UNL_u \cap UNL_v| > (1-\rho)n_v$. Hence,

$$\begin{aligned}|A| =& |UNL_u \setminus UNL_v| = |UNL_u| - |UNL_u \cap UNL_v| \\ \geq& n_u - (1-\rho)(n_u + n_v) \\ \geq& \rho \cdot n_u + (\rho - 1)n_v\end{aligned}$$

Now as per the assumption $|B| = |UNL_u \cap UNL_v| > (1-\rho)n_v$, $(1-\rho)n_v$ servers in B can publish L_1. Furthermore, if all validating servers in A publish the ledger L_1, UNL_u can receive at least $(1-\rho)n_v + \rho \cdot n_u + (\rho-1)n_v = \rho \cdot n_u$ many validations for ledger L_1. Thus UNL_u can publish the ledger L_1. On the other hand, only $(1-\rho)n_v$ validating servers in B publish for L_1. Hence, it is possible for UNL_v to publish another ledger L_2 as, $n_v - (1-\rho)n_v = \rho \cdot n_v$, resulting a fork.

Next, we look whether Equation 5.11 is sufficient to prevent fork. Assume, the validating server u accepts the ledger L_1 from its UNL and Equation 5.11 is true. The set $A_1 \cup B_1 \subseteq UNL_u$ publish the ledger L_1 i.e. $|A_1 \cup B_1| > \rho \cdot n_u$ (see Equation 5.5). To prove that Equation 5.11 is sufficient to prevent fork, it is enough to show $|B_1| > (1-\rho)n_v$, because in that case UNL_v cannot have sufficient validation for another ledger L_2.

$$\begin{aligned}|B_1| =& |A_1 \cup B_1| - |A_1| \\ \geq& |A_1 \cup B_1| - |A| \\ =& |A_1 \cup B_1| - (|UNL_u| - |UNL_u \cap UNL_v|) \\ >& |A_1 \cup B_1| - (n_u - (1-\rho)(n_u + n_v)) \\ \geq& \rho \cdot n_u - (n_u - (1-\rho)(n_u + n_v)) \\ =& (1-\rho)n_v\end{aligned}$$

Thus Equation 5.11 is necessary and sufficient to prevent any fork.
Chase and MacBrough further investigated the current version of Ripple's consensus protocol without the original byzantine accountability assumption. They concluded that in that case the *UNL* intersection has to be atleast 90% [81, Thm 8] to avoid forks. Furthermore, they provided a counterexample of liveness property [81, Sec. 4.2], where protocol can get stuck as soon as the *UNL* intersection

is less that 99%. This results essentially shows Ripple consensus protocol is far away from being a distributed BFT protocol and rigorous improvement is required for its decentralization.

5.6 Summary

We describe the Ripple consensus protocol in detail and discuss its security. Our finding shows that the parameters suggested in Ripple whitepaper [5] do not prevent forks in the system. We also discuss the conditions to prevent forks [16] and an improvement suggested by Chase and MacBrough [81].

Part III

On Privacy in Consensus-based Distributed Economic Dispatch Protocols in Smart Grid

Chapter 6

A Brief Introduction to the Smart Grid

6.1 Introduction

In this chapter, we give some background information on *Smart Grid*, its components, agents, and its security, privacy issues. Traditionally, the electricity is generated in bulk and transported via a high voltage transmission line. Afterward, it is transformed into medium voltage and finally distributed among end consumers at low voltage. In a traditional grid, the electricity cannot be stored, and generation should match the consumption. In this model, electricity mainly comes from some central generators, and generated electricity relies on the predicted consumption. Since the last decade, the traditional electricity grid has undergone some major infrastructural changes in the direction to become *smarter*. This smart grid builds a bi-directional information communication network on top of the existing energy network aiming at a more efficient, sustainable, and reliable use of energy [86, 87]. This modern electrical grid includes several advantages over the traditional electricity grid, such as real-time energy monitoring, distributed generation, energy storage, energy trading, integration of renewable energy resources, refined energy measurement, and management, etc. Each consumer is equipped with smart meters for refined consumption measurement, and data is shared with suppliers for smooth operation in the grid. However, it raises severe privacy concerns as the shared measurement data reveals private information about the users [88]. Furthermore, current energy management protocols in the smart grid are vulnerable to private information leakage [12]. In the next sections, we give an overview of the smart grid and discuss current security and privacy issues.

6.2 Smart Grid Overview

6.2.1 Definition

The term "smart grid" is not a specific state but used more like a continuous development over the "traditional grid". The definition of a smart grid varies under different circumstances and is interpreted on the basis of requirements from different stakeholders. The smart grid stakeholders can be different entities as such users, electricity network companies, electricity service companies, technology providers, traders, researchers, generation companies, regulators, government agencies [89]. First, we quote the definition given by the European Commission Task Force on Smart Grid [90]:
"A Smart Grid is an electricity network that can cost efficiently integrate the behaviour and actions of all users connected to it - generators, consumers and those that do both - in order to ensure economically efficient, sustainable power system with low losses and high levels of quality and security of supply and safety."

However, this above definition lacks some clarity as it mostly consists of requirement specifications. Later on, a more concrete definition of the smart grid was given by Jawurek [91] w.r.t. Consumer privacy and privacy-enhancing technologies. We present the following definition of smart grid inspired by Jawurek [91]:

Definition 6.2.1. (Smart Grid) The smart grid is a modern electricity grid which deploys additional interconnection and communication between agents of the traditional electricity grid with information and communication technologies (ICT) to improve quality of operation (w.r.t different aspects).

The traditional grid agents are consumers, generators, transmission operators, distribution operators, and other service providers. Whereas the quality of operation can be in different aspects like sustainability (use of more renewable energy), reliability (such as fault detection), resilience against attacks, safety, data security, privacy, etc. Additionally, the agents are interconnected with a bi-directional communication network using ICT. Some ICT components in the smart grid are smart meters, distributed energy resources, demand response techniques, privacy-preserving technologies, etc.

6.2.2 A Simple Smart Grid Model

In Figure 6.1, we depict a simple scenario of power and information flow in a smart grid. The utility providers are some agents who can do a distributed generation of electricity with different power sources (renewable, non-renewable). Then they supply it to the transmission line, and finally, the electricity is delivered among the household consumers via a data aggregator. On the consumer side, some households or small businesses are capable of producing (e.g., with solar cells) and storing (e.g., with a battery) low-voltage electricity. As these

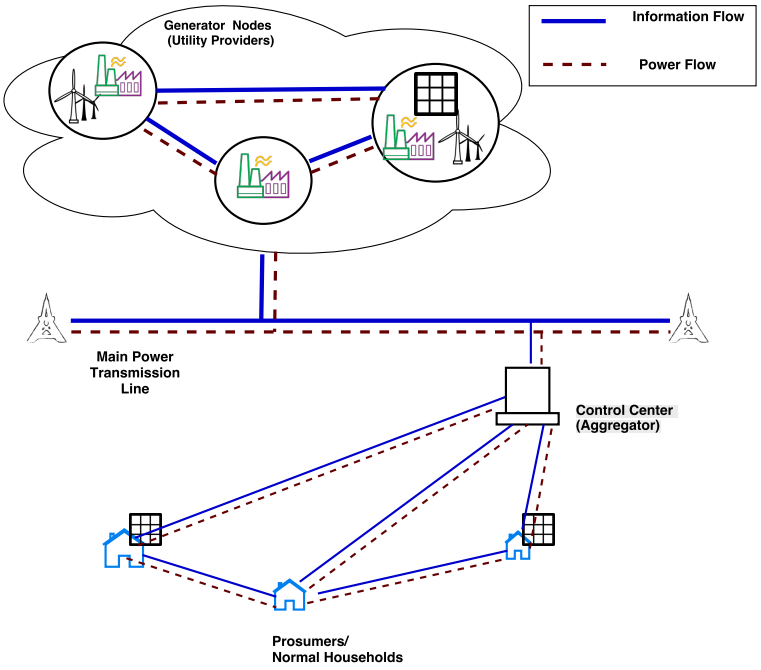

Figure 6.1: Power and Information flow in a Smart Grid

consumers can act as producers and consumers both, they are known as *prosumers*. The prosumers can sell or trade their excess electricity in the local electricity market. Every consumer is equipped with smart meters for refined consumption reading, billing purposes, and can communicate over internet protocols (IP). The role of the data aggregator is to find the total demand based on the consumption of the local prosumer network (also known as *microgrid*) and supply the needed electricity from the utility providers. On the generation side, utility providers can communicate with each other and decide how much electrical power to generate.

6.2.3 Agents in Smart Grid

Several agents are stakeholders in the smart grid infrastructure. These agents directly or indirectly need smart meter data for smart grid operation. Some main agents in the smart grid are as follows:

Consumers

Consumers are the agents who consume electricity at the supply end and pay for it. Consumers are also referred to as customers, users, households, small business enterprises, etc. If a consumer produces electricity with some Distributed

Energy Resources (DER) and it is known as a prosumer.

Utility Providers

Utility providers (UP) are the agents who are mainly are responsible for electricity generation. They can be a large corporation to a medium or small size enterprise. The network of UPs can generate electricity in a distributed manner to provide consumer demand. Different strategies of distributed energy management can be found in [8, 9]. Some examples of utility providers are Distribution System Operator (DSO) and Independent System Operators (ISO) [92]. The DSOs generate and distribute power in a medium voltage scale, whereas ISOs generate power at a high voltage scale. A utility provider has its generation units, and the cost of production depends on its cost function.

Data Aggregator

Data aggregator (also known as control center) is an agent who can monitor the aggregated data of a microgrid or network of consumers. Every consumer's smart meter is connected to the data aggregator so that it can calculate the total demand of the network. Data aggregator can act as the supplier of electricity from utility providers to consumers.

Transmission System Operator

The Transmission System Operator (TSO) is an agent who is responsible for the electricity transmission network. TSO generally operates and maintains the high voltage transmission line. TSO is also responsible for balancing demand and supply of electricity in a smart grid.

Energy Regulators

Regulators are the governing body in the smart grid industry. They have various roles, such as make policies, regulate requirements, give permits, monitor the energy market, and perform an audit on smart grid agents, etc.

Third Party

Third parties are the agents who are not directly involved in usual smart grid operations such as generation, transmission, distribution, and consumption but provide some extra services. Some examples of third parties in the smart grid are energy trading platform, consultancy services, flexibility aggregator, etc. The third parties are usually interested in user consumption data to perform analytics.

6.2.4 Smart Grid ICT Components

We describe some of the main ICT components in current smart grid operations:

Smart Meters

Smart meters are advanced electricity meters capable of bidirectional communication and can send data to other grid agents over the IP. With smart meters, a consumer can measure electricity consumption from the grid as well as the amount of electricity it produces from renewable energy resources. Additionally, smart meters can also measure different parameters like voltage level, frequency in the electricity flow. The smart meters require a collection of protocols and infrastructure as such Advance Metering Infrastructure (AMI) [93] to collect data and send data. AMI includes form physical deployments such as sensors, smart meters, monitoring devices to information communication infrastructure, data management software, etc. According to EU directive 2009/72/EC [94], the European Union encourages households for the installation of smart meters in the EU and expects 80% of consumers will be equipped with smart metering systems by 2020. Smart meter readings are collected for several reasons like load monitoring, demand response, billing, settlement, fraud detection, etc.

Distributed Energy Resources

Distributed Energy Resources (DES) are small scale decentralized generators located mainly on the distribution side. They are mainly powered by Renewable Energy Sources (RES) such as solar or wind and comes with battery storage. In consumer premises, DERs are connected with their smart meter.

Communication Networks

Smart grid agents are connected through different power and information communication networks. Power network structures are not necessarily the same as the information communication network. For example, it is possible that UPs form a fully connected graph for information communication, but their power network structure forms ring topology. On networking aspects, whereas Home Area Network (HAN) is used between the smart meter and home appliances, long-distance communication between UPs is done through Wide Area Network (WAN).

Demand Response Techniques

Demand Response (DR) strategies are implemented to balance the demand of consumers and electricity generation. As the electricity price is costly during the peak hours, a consumer might reduce its electricity bill by using heavier electric appliances at cheaper hours and lighter applications at peak hours. A TTP/DSO can offer different DR services (e.g., incentive-based, price-based) to the consumers.

Energy Management System (EMS)

Energy Management System (EMS) is used by the UPs to monitor, control, and optimize the electricity generation and transmission. One existing implementation of EMS could be with Supervisory Control And Data Acquisition (SCADA) system. SCADA system can accurately measure electrical data (e.g., voltage, frequency, current) from sensors and actuators. EMS is deployed at the microgrid level for performance improvements of DERs. EMS system and software help grid agents to perform some important functionalities like *Economic Dispatch* (ED) [95], which we will discuss in latter chapters.

For some detailed surveys in smart grid infrastructures, we suggest the readers [87, 96, 97, 98].

6.3 Security and Privacy in Smart Grid

6.3.1 Security and Privacy Challenges

The electricity grid is rapidly transforming with continuous integration of data communication infrastructure and protocols between the agents. However, it is important before large scale deployments data security and privacy issues should be identified and carefully mitigated. In the current development of the smart grid, it faces a handful number of data security and privacy challenges. McDaniel and McLaughlin identify some of the important challenges in the smart grid in [99]. The challenges are classified into four main categories:

Privacy of Agents

The private data of consumers can be revealed by analyzing consumption traces of smart metering data. It is possible to find a user's behavior of using electrical appliances over time from smart meter consumption data. For example, smart meter consumption data can tell which television channel is being watched by the user at what time. An attacker could abuse inferred information with price discrimination, use it in legal cases, marketing manipulation, financial gains, etc.

Smart Grid Protocols

Protocols for grid functionality and additional services in the smart grid should meet security and privacy requirements before deployment. Distributed protocols are replacing old centralized protocols in recent years. Such distributed protocols include economic dispatch, load balancing, optimal power flow, energy trading, etc. However, it has been shown in previous literature that many distributed protocols are not secure and reveals privacy-sensitive confidential information about agents [12, 100].

Standardization

There has been a continuous effort of standardization in smart grid infrastructure [97, 101]. The security requirements and practices in the smart grid are different than in a generic IT communication system. For example, the lifespan of a smart meter is much longer than usual mobile or computer devices. Hence, the smart meters should be compatible with security updates for a longer period.

Furthermore, availability is a critical measure; agents and services should be in operation at any point in time. However, many circulated smart grid standards do not meet security requirements and should be evaluated before large scale deployments. European Telecommunications Standards Institute (ETSI) approved the Open Smart Grid Protocol (OSGP) standard is currently deployed in the smart grid over 5 million smart meter users [102]. However, Jovanovic *et al.* in [103] showed that OSGP's authenticated encryption scheme uses some non-standard composition of RC4 and so-called "OMA digest" and successfully performed key recovery with negligible time complexity.

Cyber-Security Threats in Infrastructure

Cyber-security attacks can be costly for smart grid infrastructure [104]. Attackers can hack remotely online smart grid services and cause a blackout on a massive scale. Furthermore, components of infrastructure such as smart meters should be tamper-resistant to prevent billing fraud or false data injection attacks. One example of a cyber attack was the Ukrainian power grid attack in 2015, which compromised IT infrastructure and interrupted power grid operation [105].
In the next sections, we discuss the state of the art privacy preservation approaches in the smart grid in relevance to our work.

6.3.2 Privacy Policies and Data Protection Laws

The definition of privacy has a different meaning from multiple standpoints. Its definition has been discussed expensively among legal, social, physiology, and computer science researchers. The very first definition of privacy can be found in the 1890 law review article by Warren and Brandies [106], where they defined it as "right to be let alone". Alan Westin in 1967 [107] defined privacy as one's fundamental right to self-determine what information about himself to be transferred to others in what circumstances. Current European data protection laws follow from Westin's definition about which encompasses about right to determine which personal information to give. The privacy laws and regulations states set rules under which conditions personal data can be collected and used. For instance, the German data protection law describes that the personal data can only be collected under the user's *consent* and for some specific use. The protection of private data in current smart grid deployment is mostly achieved through privacy policies from the regulators. Regulators can frequently perform audits to check if data collecting agents are following the policies. The General

Data Protection Regulation (EU) 2016/679 (GDPR) [1] has been implemented in the EU in 2018. The GDPR framework includes "Data protection by design and by default" (Art. 25 GDPR) and is applicable in the context of data aggregator collecting smart meter data.

6.3.3 Privacy Enhancing Technologies

The privacy policies do not inherently ensure the privacy of agents. For example, some legislations might require to pseudonymize the metering data, and the smart meters are deployed with static pseudonyms. However, the aggregator might able to link the smart meters with pseudonyms by matching user's attributes with one pseudonym's attribute, thus resulting in a privacy violation. In contrast, the smart grid needs Privacy Enhancing Technologies (PET) to support the privacy policies from a technical standpoint. Where privacy laws can protect the agents and give a legal basis (e.g., lawsuit) against privacy violators, PETs should prevent privacy violations from happening beforehand. In literature, different PETs have been used to propose secure smart grid protocols to perform data aggregation, electricity billing, economic dispatch, electricity trading, load scheduling, optimal power flow, demand response, etc. On the privacy perspective, we define three roles of grid agents in a generic smart grid protocol as Data Owners (DO), Computing Agents (CA), and Output Parties (OP). An agent in the role of DO is the owner of its private data. The task of CA is to perform the necessary computation. Finally, the role OP is to receive the result of the calculation. It possible that one agent might act in multiple roles in some protocol execution. For instance, UPs can operate in three types of functions as DO, CA, and OP in a distributed economic dispatch protocol [13]. In local energy trading, the prosumers act as both DO and OP, and trading platform acts as CA [108]. In the case of most data aggregation protocols, the consumer is DO, the aggregator is OP, and CA can be consumers themselves or aggregator or a trusted third party. We discuss some PETs used in smart grid protocols as follows:

Anonymization

Anonymization or pseudonymization techniques are extensively used in current smart grid protocols such as aggregation. The goal is that CA can perform computation on data from all DOs, but specific smart meter data and its originator should be unlinkable. Where pseudonymization replaces the identity of DO's with a *pseudonym*, anonymization requires complete unlinkability. Pfitzmann and Hansen provide a concrete set of definitions about anonymity, pseudonymity, and unlinkability in [109]. Some pseudonymization methods can be ineffective depending on the features and frequency of the data, as some classification based attacks could reveal the identity of the originator. In [91], Jawurek *et al.* showed de-pseudonymization of the consumer metering data. They can link user profiles with its pseudonym by unusual pattern detection

[1] `https://gdpr-info.eu/`

and machine learning techniques. Anonymization techniques are not suitable in electricity billing protocols as the electricity provider needs to send the bill to the consumer.

Perturbation

Perturbation based-approaches can be used in aggregation, economic dispatch protocols. In perturbation based protocols, DO can add some random noise in its metering data before sending it to the CA; OP can find computed value from CA with some marginal error without knowing any private data. In [110], authors achieved differential privacy with perturbation for generic distributed optimization protocols in the smart grid. In [111], authors performed centralized privacy-persevering economic dispatch computation with the perturbation method. Perturbation techniques are generally not applicable in billing protocols as its bills are expected to be exact.

Cryptographic Computation

Cryptographic computation techniques like SMC or homomorphic encryption (HE) can be integrated into smart grid protocols to perform secure computations. DO can secret share or encrypt their private data for computation. These protocols ensure that CA can not deduce private DO information while performing computation from secret shares or on ciphertexts (HE).
Li *et al.* [112] proposed some aggregation protocol where DOs encrypts its data items with paillier encryption, routes, and aggregates through a minimal spanning tree to a root OA. Kursawe *et al.* [113] proposed several smart meter aggregation and comparison protocols. Their protocols use unidirectional communication links and techniques like secret sharing and Diffie-Hellman key exchange to bind the meter measurements before aggregated by CA. Erkin *et al.* [114] presented a system where DO uses paillier encryption to send the data to CA for aggregation. In [12, 13], we have proposed SMC-based distributed privacy-preserving protocols to solve the economic dispatch problem, which we will discuss in latter chapters. Abidin *et al.* proposed a SMC-based energy trading protocol for local electricity market in [108]. In their protocol, prosumers secret share their bids in a decentralized trading platform to trade their excess electricity. The trading platform controlled by independent agencies can do the bid selection and trading price calculation. They could able to compute market tasks with 2500 bids in less than 4 minutes. One of the other works include Li *et al.* [115], where they presented a privacy-preserving demand response scheme with homomorphic encryption and adaptive key evolution.

Verifiable Computation

Verifiable computation is used when the CA is untrusted, and CA needs to provide proof to the DO that the required computation has been performed as it should be. These type of protocols ensures the integrity of the computational

results. Verifiable computation is a common tool building privacy-preserving billing protocols. In [91], a billing scheme is shown where smart meter users commit their consumption data with Pedersen commitment. The protocol gets tariff from the supplier and can generate a proof for correct bill calculation with Zero-Knowledge Proof (ZKP). On the other hand, the supplier can verify the correctness of the bill calculation from the proof and commitments of consumption data.

Protected Module Architectures

Protected Module Architectures (PMA) are hardware security architectures for trusted computation. PMA supports a trusted CA to securely execute some software modules in an isolated environment, even if CA gets infected. A remote DO can use PDA and collect trust evidence that CA is still in a trustworthy state. Intel Software Guard Extensions (SGX) is an implementation of PDA where one can run private protocols in isolated memory containers called *enclaves*. These hardware enforceable containers are protected from any processes outside the enclave [116]. In [117], authors form KU Leuven presented a prototype of a smart metering security infrastructure with *Sancus*, a PMA implementation for low-powered microcontrollers.
For a detailed survey in privacy-enhancing technologies in the smart grid, we recommend the readers [88].

6.4 Summary

In this chapter, we have outlined the necessary background on the smart grid, its components, and agents. We have provided an outline of different security and privacy concerns and the current use of privacy-enhancing technologies in smart grid protocols. In the next chapters, we will introduce the economic dispatch (ED) problem and develop our privacy solutions for consensus-based distributed ED protocols for the smart grid.

Chapter 7

Economic Dispatch (ED) Problem

7.1 Introduction

In the last chapter, we gave an overview of the smart grid, its security, and privacy challenges. A good number of ideas have been proposed to solve smart metering privacy problems such as aggregation and billing at the microgrid level [88]. However, privacy issues in the area of energy management protocols have received a relatively modest amount of research interest. In this chapter, we introduce a crucial smart grid energy management problem called *economic dispatch* (ED), identify its privacy problems, and survey existing literature.

7.2 What is Economic Dispatch?

Economic dispatch (ED) is one of the fundamental optimization problems in the electrical power grid. Over the past few decades, it has been an important topic of research by itself in the power grid community [95, 118, 119, 120]. Its goal is to find the optimal power output of all generator nodes in the grid that meets the power demand of the consumers at the lowest possible cost.
To give the readers a concrete example, we recall some of the smart grid agents introduced in the last chapter: a set of utility providers (UP), a data aggregator, and a set of consumers. Each UP has its generation facility, and together they form a generation network. The job of the data aggregator is to act as an intermediary agent to supply electricity from the generation network to all consumer loads at the lowest possible cost. To perform this, the aggregator has to know what is the total electricity demand of all consumers. Secondly, the aggregator needs different information from the UPs. For instance, some essential information is the generation limits, the amount of electricity each generator can produce at what cost, etc. After getting related data, the aggregator runs

an optimization algorithm and decides on the amount of energy he would buy from different UPs to deliver consumer demand at a minimum cost.
The ED problem considers two system constraints:
i) Total energy generation must be equal to total demand.
ii) Utility providers must produce electricity within its generation capacity.
Hence, the ED solution gives a power output combination of all UPs, which offers minimum total operating cost while maintaining generator and demand constraints. In traditional electricity grids, a central operator is responsible for meeting the customer's demand by coordinating the power production of a group of UPs. It solves the ED problem centrally and attempts to minimize the global cost. However, for smart power grids, decentralized solutions are becoming increasingly popular. Note that in this Chapter, Chapter 8, and Chapter 9, the terms utility providers, parties, and generator nodes are synonymously used.

7.2.1 Centralized Solutions for Traditional Grid

Traditionally, the ED problem is usually solved in a centralized fashion. One agent (e.g., data aggregator or a third party) acts as a trusted entity. The trusted agent collects different parameters and associated constraints (e.g., generator limits, cost function) from all UPs. It performs optimization calculations and sends the optimal solution to every generator. The trusted agent can use different optimization algorithms to solve the ED problem. While numerical methods like lambda-iteration or gradient search [121] are conventionally used, use of more computationally intensive techniques like particle swarm optimization [119] and genetic algorithms [118] have also been investigated in the literature. However, a small change in the smart grid or failure of the trusted leader requires a drastic change in the centralized scheme. As a result, these centralized schemes fail to achieve many requirements of the modern smart grid, which needs a robust and reliable infrastructure.

7.2.2 Consensus-based ED Solutions for Smart Grid

Recently, researchers are applying distributed consensus algorithms to solve the ED problem to overcome the challenges of centralized schemes [120, 8, 9].
In [120], the authors adopted the equal *incremental cost* (IC) (also known as *marginal cost*) optimization criterion with lambda-iteration method and the IC of each generator is chosen as the consensus variable. IC is the increase in total cost resulting from an increase in power generation. The proof of the same incremental cost criterion (i.e., when each generator has the same IC values, we have an optimal solution of the ED problem) can be found in [121]. In [120], each generator sends its own IC to its neighbor, and the proposed consensus algorithm drives all individual IC to a common value. Importantly, the mismatch between demand and the total power generated is fed back to the consensus algorithm to meet the demand constraint. However, their algorithm is not completely distributed because a trusted agent has to be deployed to collect the

power generated by each generator to calculate the total mismatch.
In [122], the authors took a different approach to solve the ED problem. They considered the total power generated by all UPs as a linear piece-wise continuous function of IC. In their decentralized approach, the ED problem is solvable with a *ratio consensus algorithm* if the demand lies in one of the linear segments. However, the ratio consensus algorithm reveals confidential consensus parameters and generator constraints.
Yang *et al.* in [8] use similar IC criteria as in [120]. The authors in [8] considered a strongly connected network. They claimed that as opposed to [120], in their algorithm, every generator does not need to know the cost function parameters of the other generators. In this algorithm, generators agree on the optimal IC, and estimation of mismatch between demand and total power are obtained collectively through local interactions between the generators. Moreover, their solution is distributed as no trusted agent is needed to collect all the power generated by every UPs.
Binetti *et al.* [9] proposed a consensus-based distributed ED protocol based on an auction mechanism. Unlike IC as the consensus variable in [8], the participants in Binetti's ED protocol [9] agree on winning bids and winning parties. In contrast to [8], which considers the quadratic cost function of UPs, Binetti's protocol can solve ED even with non-convex cost functions.

7.3 ED Problem Formulation

We assume that n number of UPs $(P_1, \cdots, P_n)$ are forming a generation network to deliver consumer electricity demand of D units. Suppose, P_i generates x_i units of power, and we denote its cost function for power generation by C_i. Hence, P_i can produce x_i units of power with a cost $C_i(x_i)$. We also assume generation limits of P_i are from $\underline{x_i}$ to $\overline{x_i}$ i.e., P_i has to produce at least $\underline{x_i}$ units of power to be in operation and can produce up to $\overline{x_i}$ units. The produced power x_i is generally represented in MWh or KWh depending on the generator capacity. MWh is commonly used wholesale electricity trading market, whereas in a microgrid level it is expressed in KWh. In a n node generation network, the total cost of operation C_{total} can be formulated as:

$$C_{total} = \sum_{i=1}^{m} C_i(x_i) \tag{7.1}$$

Definition 7.3.1. *(Economic Dispatch (ED))* In a n node generation network, the ED protocol optimizes a combination of x_i's such that C_{total} is minimum while meeting two system constraints:

- **Demand Constraint:** The total produced power meet the customer demand D as:

$$D = \sum_{i=1}^{m} x_i \tag{7.2}$$

- **Generator Constraint:** P_i produce power within its generator limits:

$$\underline{x_i} \leq x_i \leq \overline{x_i} \tag{7.3}$$

Cost Function

In the ED problem, the cost function of P_i is usually represented by a quadratic function of the power output x_i [8]:

$$C_i(x_i) = a_i x_i^2 + b_i x_i + c_i \tag{7.4}$$

where a_i, b_i and c_i are the cost function parameters of party P_i.

Incremental Cost (IC)

Incremental cost or marginal cost of a generator is the cost of producing one unit of extra power and is linear for quadratic cost function:

$$\lambda_i = \frac{dC_i(x_i)}{dx_i} = 2a_i x_i + b_i \tag{7.5}$$

Non-convex ED Problem

A non-convex ED problem is considered more practical in smart grid systems as the cost function includes valve point effect, multiple fuel option and prohibited operating zones [9]. An additional sinusoidal term is included in the cost function for a non-convex ED problem as:

$$C_i(x_i) = a_i x_i^2 + b_i x_i + c_i + |d_i \sin(e_i(\underline{x_i} - x_i))| \tag{7.6}$$

7.4 Privacy in ED

7.4.1 Attacker Model

The ED protocol attackers can be internal, as well as external. External attackers can be completely malicious. Internal attackers, such as UPs might choose to behave maliciously. For instance, a UP may modify their input data to gain maximum profit or collude with other UPs to outplay their competitors. However, the electricity market is highly regulated. For smart grid participants behaving maliciously might be a risky endeavor, since a convicted cheater might face a permanent ban from the energy market by the regulatory board. Therefore, the internal attackers in ED protocols are assumed as semi-honest entities.

7.4.2 Privacy Goals

In ED problem, the UP's output power, cost function parameters, and minimum/ maximum output power are privacy-sensitive data [12].

Output Power (x_i):

Revealing output power to other UPs may harm the business model of the utility company. In a competitive market, if the power output gets leaked to other competitors, competitors might generate more power during peak hours, and the UP might get outplayed from the energy market.

Cost Function Parameters (a_i, b_i, c_i):

The cost function of a UP is confidential and critical business information. Hence, the cost function parameters of the generators are privacy-sensitive. By knowing the operational cost of the competitors, a UP can set up a target to reduce its own operational cost, and establish itself as the least-cost utility provider in the market. Moreover, sharing one UP's cost function might further reveal salary-related information of its employees, business process strategies, etc. Revealing the parameters (a_i, b_i, c_i) of a UP P_i, a competitor will get to know UP's cost function of P_i.

Minimum ($\underline{x_i}$) and Maximum ($\overline{x_i}$) Power Output:

Individual generator constraints are privacy-sensitive. A UP might want to keep its generating capacity private. A competitor can adjust its power generation based on other generator's output capacity and the demand curve to be the key player in the market. Furthermore, prior knowledge of generator ratings can lead to system attacks in ED protocols [123].

7.5 A Survey of Privacy-preserving ED

There are only a few works on security and privacy of ED protocols (or similar energy management problems like *optimal power flow* (OPF) [124] or *load scheduling* (LS) [125]) in the current literature. In this section, we give a comprehensive survey of related works. ED and OPF are a similar type of energy optimization problem. Whereas OPF considers line flow limits and constraints in a transmission network, ED calculation does not consider power losses and transmission constraints. Furthermore, OPF models are approximated to linear models and not accurate after certain operating points [126]. On the contrary, LS problem [125] aims to optimize energy distribution across different facilities at a low voltage level.

At the time of writing, we list previous security and privacy related works in ED protocols in Table 7.1. We further classify the related works based on centralized and distributed as follows:

Centralized: State of the art centralized privacy solutions for ED are mostly perturbation based approaches. The participants add random noise to their input data and send it to a semi-trusted control center for calculation. The perturbation approach aims to provide a trade-off between the level of achievable privacy and solution accuracy. Early work on privacy in the OPF problem can

Paper	Year	Problem	Model	Method	Optimization
Borden *et al.*[124]	2012	OPF	Central	Perturbation	Linear Programming
Borden *et al.*[127]	2013	OPF	Central	Perturbation	Quadratic Function
Rottondi *et al.*[125]	2013	LS	Distributed	SMC	Linear Optimization
Huang *et al.*[110]	2015	Generic	Distributed	Perturbation	Quadratic Function
Mandal [12]	2016	ED	Distributed	SMC	Quadratic Function
Wu *et al.*[128]	2016	OPF	Central	Perturbation	Non-convex
Shelar *et al.*[123]	2017	ED	Central	System Attack	Quadratic Function
Zhao *et al.*[129]	2017	LS	Distributed	Perturbation	Quadratic Function
Zhao *et al.*[100]	2017	ED	Distributed	Injection Attack	Quadratic Function
Yang *et al.*[130]	2017	OPF	Central	Perturbation	Quadratic Function
Zhao *et al.*[131]	2018	ED	Distributed	Perturbation	Linear Approximation
Liu *et al.*[132]	2018	OPF	Distributed	Perturbation	Quadratic Function
Mandal *et al.*[13]	2018	ED	Distributed	SMC	Non-convex Auction-based
Sarker *et al.*[133]	2018	ED	Central	Cloud Framework	Linear Programming
Wu *et al.*[111]	2018	ED	Central	Perturbation	Linear Programming
Lu *et al.*[134]	2018	Generic	Distributed	HE	Gradient-based
Wu *et al.*[135]	2019	ED	Central	Perturbation	Linear Programming

Table 7.1: Security and Privacy in ED by Work

be found in [124, 127]. In 2012, Borden *et al.* in [124] proposed a privacy solution to preserve power system parameters in a centralized linear OPF problem by obfuscating the transmitted data. In [127], authors extended their solution of [124] for quadratic OPF problem. Wu *et al.* in [128] has shown a similar masking approach for the non-convex OPF problem. Some similar perturbation based approaches have been applied for the centralized ED problem [135, 130]. Yang *et al.* [130] provided a centralized solution for optimal power flow while introducing Gaussian noise from the parties, and achieved differential privacy. In 2018, Sarkar *et al.* [133] provided a general framework on privacy-preserving centralized outsourcing framework for perturbation based ED with linear programming optimization. Some orthogonal work related to ED security is shown by Shelar *et al.* in [123], where authors performed a semantics-based attack on the tools used by the control centers. They demonstrated the attack by manipulating control parameters in insecure energy management software for centralized ED calculation. The attack essentially gives false ED solutions (e.g., power output exceeding generator constraints), which violates the safety of the smart grid infrastructure.

Distributed: Even though a handful of consensus-based distributed solutions for ED have been proposed, academic efforts for privacy-preserving distributed ED is still in the early development. As far we are aware, our work in [12] was the first contribution towards privacy-preserving ED solution. In [12], we gave an attack on the ED protocol from Yang *et al.* [8] and proposed a privacy-preserving distributed protocol in the information-theoretic model with a quadratic cost function. We will discuss the work [12] in detail in Chapter 8.

Liu *et al.* introduced a distributed OPF protocol with privacy leakage mitigation using a stochastic noise method for a radial topology [132]. Huang *et al.* in [110] has given a generic distributed algorithm for constraint minimizing of a quadratic cost function and achieves differential privacy. Zhao *et al.* proposed a consensus-based load scheduling algorithm by a zero-sum and noise reduction technique [129] for privacy preservation. Furthermore, in [129], the authors considered the LS problem as quadratic and aim to maximize a social welfare function consisting of a profit function and cost function. Related to the LS problem, Rottondi *et al.* also gave a privacy-friendly solution for distributed LS considering a linear function by using SMC. In [100], the authors showed a false data injection attack on consensus-based ED protocols in a distributed setting.

In [13], we proposed a privacy-preserving distributed ED protocol for non-convex cost function using SMC. In chapter 8, we will discuss our work in detail. Further ED related work includes [131], where authors gave a leader based distributed ED protocol with linear approximation and perturbation. More recently, Lu *et at.* [134] has given a generic gradient-based constraint optimization with homomorphic encryption (HE).

7.6 Summary

We introduced the necessary ED background for our work, privacy goals, and related works. In the next two chapters, we construct two distributed privacy solutions for the ED problem. Our first solution is based on [8] and the other one on [9].

Chapter 8

Privacy-preserving ED Protocol I

8.1 Introduction

In this chapter, we construct our first privacy solution for distributed economic dispatch (ED). We first describe a well-established distributed ED protocol in the smart grid community from Yang *et al.* [8]. Then we propose the privacy-preserving ED (*PPED*) protocol for distributed ED, which is based on Yang *et al.*'s protocol. The presented work here is previously published in FNSS'16 [12]. To the best of our knowledge, this work was the first attempt to solve privacy problems in the distributed ED protocols. The contributions of this work are as follows:

- We identify which information should be private in the ED calculation (defined in the Chapter 7, section 7.4).
- We analyze the security of distributed ED protocol from Yang *et al.* [8]. We show how privacy-sensitive information is leaking even under a simple semi-honest attacker model without any collusion.
- We provide a solution to privacy problems in distributed ED by adding a privacy layer with a secure sum protocol on top of Yang *et al.* [8].
- We propose an information-theoretic privacy model for this type of protocols and give a security proof for the proposed protocol in our model.
- We improve upon the version presented in FNSS'16 [12], by achieving better communication complexity.

Notation	Description
G	Network graph of the protocol participants
V, E	Set of vertices and edges in G
n	Total number of generator nodes in the network G
i, j	Different UP or generator nodes in network
t	Discrete time step for each round
$x_i(t)$	Output power estimate of node i at round t
C_i	Cost function of node i
$\lambda_i(t)$	Incremental cost of node i at time step t
a_i, b_i, c_i	Cost function parameters
$\alpha_i, \beta_i, \gamma_i$	Cost function parameters used in [8]
D_i	Local power demand for node i
D	Total power demand in the network ($D = \sum_{i=1}^{n} D_i$)
$\underline{x_i}$, $\overline{x_i}$	Minimum and maximum output power limit of generator i
$\overline{y_i}(t)$	Power mismatch of node i at round t
p_{ij}, q_{ij}	Different elements of admittance matrix P and Q of the network
N_i^+, N_i^-	In and out neighbors of node i
ϵ	Very small public constant

Table 8.1: Nomenclature

8.2 Yang *et al.*'s Consensus-based ED [8]

The existing state of the art distributed ED solutions in [120] [8] use similar structures. The Yang *et al.*'s ED protocol [8] is an iterative algorithm, uses *incremental cost* (IC) as the consensus variable and considers quadratic cost function.

8.2.1 Incremental Cost Criteria and Notations

Recall the ED problem defined in the Chapter 7, section 7.3. We consider, a group of generator nodes $V = \{1, \ldots, n\}$ jointly aims to solve the ED problem. At time step t, we assume, the estimated power production of the generator node $i \in V$ is $x_i(t)$. We also consider, every generator node $i \in V$ has a quadratic cost function C_i. For instance, the cost of power production of node i at time step t is $C_i(x_i(t))$. The total customer demand is denoted by D, which the network V must produce and supply. Moreover, Yang *et al.*'s ED protocol considers a local customer demand D_i for initialization of node i such that $D = \sum_{i=1}^{n} D_i$. The *incremental cost* IC is the increase in total cost resulting from an increase in power generation. We define the IC of generator node i at time step t is $\lambda_i(t)$.

The *equal incremental cost criterion* [121] met when every node has equal incremental cost, resulting in an optimal solution for the ED problem. The notation used in the chapter is given in Table 8.1. The incremental cost (IC) is used as the consensus variable and every nodes aims to reach the same IC after rounds of communication and computation.

8.2.2 System Model

In [8], authors assumed a strongly connected network topology as a directed graph $G = (V, E)$. The set of vertices $V = \{1, \ldots, n\}$ represent the generator nodes (or utility providers) of the network and the set of edges $E \subseteq V \times V$ represent the communication structure between the nodes. Strongly connected means, there exists a path between any pair of two vertices in the directed graph. A direct edge from i to j is denoted by an ordered pair $(i, j) \in E$ and means that a node j can receive information from i. The in-neighbors and out-neighbors of i^{th} node are represented by $N_i^+ = \{j \in V | (j, i) \in E\}$ and $N_i^- = \{j \in V | (i, j) \in E\}$ respectively. A node can receive information from in-neighbors and send information to out-neighbors. As each node can know its own state information, each vertex belongs to both its in-neighbors and out-neighbors ($i \in N_i^+$ and $i \in N_i^-$).

8.2.3 Description of the Incremental Cost Consensus Algorithm (Yang *et al.* [8])

Here, two matrices are defined as $P, Q \in \mathbb{R}^{n*n}$ based on the topology of the graph G. Let, p_{ij} and q_{ij} be the elements of matrices P and Q respectively. All the elements of P and Q are public. They are defined as:

$$p_{ij} = \begin{cases} \frac{1}{|N_i^+|} & \text{if } j \in N_i^+ \\ 0 & \text{otherwise} \end{cases}$$

$$q_{ij} = \begin{cases} \frac{1}{|N_i^-|} & \text{if } i \in N_j^- \\ 0 & \text{otherwise} \end{cases}$$

The standard cost function used for ED calculation is quadratic as shown in (7.4). In [8], a slightly different quadratic cost function is being used (quadratic convex).

$$C_i(x_i) = \frac{(x_i - \alpha_i)^2}{2\beta_i} + \gamma_i \tag{8.1}$$

Where $\alpha_i \leq 0$, $\beta_i > 0$ and $\gamma_i \leq 0$. However, the cost functions (7.4) and (8.1) are basically equivalent if we replace $\alpha_i = -(b_i)/(2a_i)$, $\beta_i = 1/2a_i$ and $\gamma_i = c_i - (b_i^2)/(4a_i)$. The IC of node i at a discrete time index t can be formulated as:

$$\lambda_i(t) = \frac{x_i(t) - \alpha_i}{\beta_i}$$

– **Initialization:** D_i is the local demand associated with the node i. Hence, the total demand is $D = \sum_{i \in V} D_i$.
Initially, $\forall i \in V$, we have:

$$x_i(0) = \begin{cases} \overline{x_i}, & \text{if } \overline{x_i} < D_i \\ D_i, & \text{if } \underline{x_i} \leq D_i \leq \overline{x_i} \\ \underline{x_i}, & \text{if } D_i < \underline{x_i} \end{cases}$$

$$\lambda_i(0) = \frac{x_i(0) - \alpha_i}{\beta_i}$$

$$y_i(0) = D_i - x_i(0)$$

– **Main Algorithm:** At round $t+1$, the variables get updated as follows:

$$\lambda_i(t+1) = \sum_{j \in N_i^+} p_{ij}\lambda_j(t) + \epsilon y_i(t)$$

$$x_i(t+1) = \beta_i \lambda_i(t+1) + \alpha_i$$

$$y_i(t+1) = \sum_{j \in N_i^+} q_{ij} y_j(t) - (x_i(t+1) - x_i(t))$$

At each round, every node updates its own IC $\lambda_i(t)$ from the IC values of the previous round received from its in-neighbors N_i^+. Then, each node updates its output power $x_i(t)$ from the updated IC. Subsequently, i calculates the mismatch $y_i(t)$ and forwards to its out-neighbors N_i^- for the next round. ϵ is a positive scalar and controls the convergence speed. Suppose, λ^* is the optimal value for IC and ${x_i}^*$ is the optimal power output of node i for ED solution. Yang *et al.* claimed that if ϵ is sufficiently small the algorithm converges to the ED solution i.e.,

$\lambda_i(t) \to \lambda^*$, $x_i(t) \to {x_i}^*$, $y_i(t) \to 0$ as $t \to \infty, \forall i \in V$ [8, Thm. 2 and 3].

8.3 Attack on Yang *et al.*'s ED Protocol

In this section, we show an attack to the Yang *et al.*'s protocol described in the previous section.

8.3.1 Attacker Model

We assume the attacker i is a non-colluding semi-honest node in the network, i.e. i strictly follows the protocol but it may analyse the messages exchanged during the execution of the protocol to gain additional information. We also assume that the attacker knows the local demand D_j of any other node j in the network. This assumption is realistic as the local demand D_j is basically the aggregated demand from the consumers of j and can be a public information. The privacy goals in distributed ED is given in section 7.4.2. An attack will be successful if the attacker achieves full knowledge about output power, generator constraints, and any cost function parameter corresponding to another node.

8.3.2 Privacy-sensitive Data Leakage

Let us assume that the messages are exchanged between two nodes, i (attacker) and j (another node) where $i \in N_j^-$ and $N_j^+ \subseteq N_i^+$.

– **Leakage of Output Power (x_j)**: At $t = 0$, i receives $\lambda_j(0)$ and $y_j(0)$. As i knows the local power demand D_j, i can simply get the value of $x_j(0)$. Then, i can find out the output power estimate $x_j(t)$ at every round as i can get $\sum_{k \in N_j^+} q_{jk} y_k(t)$ from each k in N_j^+ at round t and $y_j(t+1)$ from j at round $t+1$:

$$x_j(t+1) = \sum_{k \in N_j^+} q_{jk} y_k(t) - y_j(t+1) + x_j(t)$$

If the protocol ends at round T, $x_j(T)$ is the final power output estimate for node j. This can be easily estimated by node i.

– **Leakage of Cost Function Parameters (a_j, b_j)**: i knows values of $\lambda_j(t)$ and $x_j(t)$, from two rounds t and $t+1$:

$$x_j(t) = \beta_j \lambda_j(t) + \alpha_j$$
$$x_j(t+1) = \beta_j \lambda_j(t+1) + \alpha_j$$

Solving two linearly independent equations, node i can find the values of α_j and β_j which will give values of a_j and b_j.

– **Leakage of Generator Constraint Parameters ($\underline{x_j}$ and $\overline{x_j}$)**: At initial round $t = 0$, i observes $x_j(0)$, so i might get one of the values of $\underline{x_j}$ or $\overline{x_j}$.

8.4 Privacy-preserving ED (PPED) Protocol

To prevent the privacy leakage, we present our PPED protocol which uses a privacy layer in each round using a secure sum protocol. We can use the consensus-based algorithm proposed by Yang et al. [8] described in section 8.2.3 as the basis. The secure sum protocol used in PPED is similar to the $n - private$ protocol for summation presented by Benaloh in [41] but uses $(n-1)$ partitions instead of n.

8.4.1 System Model

We consider a complete synchronous network $G = (V, E)$ with n nodes and a secure and reliable point to point communication channel between every node (no eavesdropping). As G is a complete graph, every element in P and Q matrix will be $\frac{1}{n}$.
A practical assumption would be that all numerical values we want to compute are fixed point values for ED algorithms. We can multiply any fixed point elements with a suitable constant and convert them into integers. In Step 2 (next section), we convert $\lambda_i(t)$ and $y_i(t)$ to integer and in Step 7 we convert these values back to the fixed point domain.

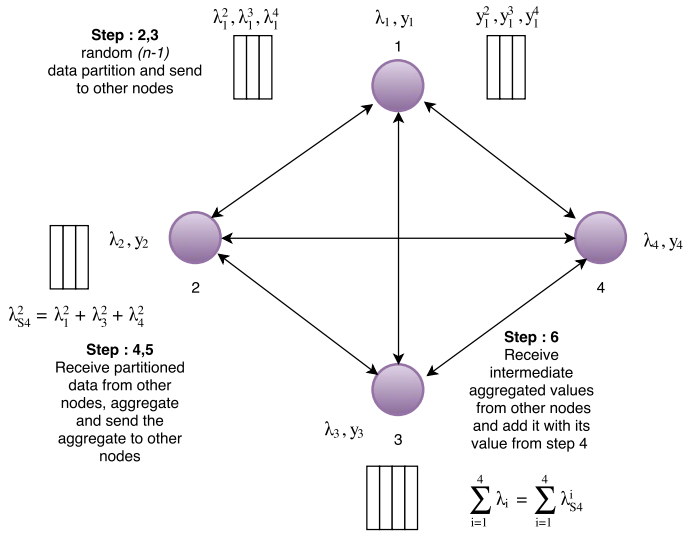

Figure 8.1: PPED protocol for $n = 4$

8.4.2 PPED protocol

The diagram of our PPED protocol with 4 nodes is illustrated in figure 8.1. The protocol is outlined as follows:

– **Step 1**: Initialization of every node at $t = 0$:

$$x_i(0) = \begin{cases} \overline{x_i}, & \text{if} \quad \overline{x_i} < D_i \\ D_i, & \text{if} \quad \underline{x_i} \leq D_i \leq \overline{x_i} \\ \underline{x_i}, & \text{if} \quad D_i < \underline{x_i} \end{cases} \quad \forall i \in V$$

$$\lambda_i(0) = \frac{x_i(0) - \alpha_i}{\beta_i}$$

$$y_i(0) = D_i - x_i(0)$$

– **Step 2**: In step 2 to step 6 we map $\lambda_i(t)$ and $y_i(t)$ values to integer values and map it back in step 7. We consider $\mathbb{Z}_M$ as the additive group of integers from 0 to $M-1$ (M is a large integer such that $M > \sum_{i=1}^{n} \lambda_i(t)$ and $M > \sum_{i=1}^{n} y_i(t)$). Every node i, chooses $(n-1)$ numbers independently with uniform distribution from $\mathbb{Z}_M$, such that their sum (in modulo M) is equal to $\lambda_i(t)$. The same is done for breaking $y_i(t)$ into $n-1$ parts where each segment is allotted for a specific node. For example, i creates the

segment $y_i^{(j)}(t)$ for the node j. All calculations are done in modulo M.

$$\lambda_i(t) = \sum_{\forall j \in V, j \neq i} \lambda_i^{(j)}(t)$$

$$y_i(t) = \sum_{\forall j \in V, j \neq i} y_i^{(j)}(t)$$

- **Step 3**: Every node i sends segments $\lambda_i^{(j)}(t)$ and $y_i^{(j)}(t)$ to the respective j in the network. We can use some token to distinguish between $\lambda_i(t)$ and $y_i(t)$ segment values.
- **Step 4**: Each node i will receive a total of $n-1$ data segments for $\lambda(t)$ and $y(t)$ respectively from the other $n-1$ nodes in the graph. All the nodes add all their received segments for $\lambda(t)$ and $y(t)$ separately. At node i, the total received sums $\lambda_{S4}^i(t)$ and $y_{S4}^i(t)$ is calculated as follows:

$$\lambda_{S4}^i(t) = \sum_{\forall j \in V, j \neq i} \lambda_j^{(i)}(t)$$

$$y_{S4}^i(t) = \sum_{\forall j \in V, j \neq i} y_j^{(i)}(t)$$

- **Step 5**: Every node i sends $\lambda_{S4}^i(t)$ and $y_{S4}^i(t)$ to the remaining $n-1$ nodes in the network.
- **Step 6**: Every node i adds all received $\lambda_{S4}^j(t)$'s from the other $n-1$ nodes and its own $\lambda_{S4}^i(t)$. Hence, every node gets sum value without knowing the individual inputs:

$$\sum_{\forall i \in V} \lambda_i(t) = \sum_{\forall j \in V, j \neq i} \lambda_{S4}^j(t) + \lambda_{S4}^i(t)$$

  Do the same for $y(t)$.
- **Step 7**: Node i finds:

$$\lambda_i(t+1) = \frac{1}{n} \sum_{\forall i \in V} \lambda_i(t) + \epsilon y_i(t)$$

- **Step 8**: For all nodes i, the power output is calculated as:

$$x_i(t+1) = \beta_i \lambda_i(t+1) + \alpha_i$$

- **Step 9**: For all nodes i, the power difference is compared as:

$$y_i(t+1) = \frac{1}{n} \sum_{\forall i \in V} y_i(t) - (x_i(t+1) - x_i(t))$$

– **Step 10**: Check:
if

$$\forall i \in V, y_i(t+1) \approx 0$$

then ED solution found **break**

else t = t+1, Repeat $\forall i \in V$ Step 2 to Step 10.

Remark. *(Correctness of PPED)* If Yang *et al.*'s ED protocol [8] is correct, the PPED protocol is also correct.

Proof. Correctness of Yang *et al.*'s ED protocol is shown in [8, Thm 2 and 3]. Hence the correctness of PPED protocol follows from, for any round for $t = \{0, 1, \ldots, T\}$, $\sum_{\forall i \in V} \lambda_i(t) = \sum_{\forall j \in V, j \neq i} \lambda^j_{S4}(t) + \lambda^i_{S4}(t)$ and $\sum_{\forall i \in V} y_i(t) = \sum_{\forall j \in V, j \neq i} y^j_{S4}(t) + y^i_{S4}(t)$. □

8.5 Security Analysis of PPED Protocol

We consider a non-colluding semi-honest adversary for the security analysis. The privacy model proposed in this section is information-theoretic and inspired from [136, 32]. To formally analyze the security of PPED protocol, we define the security model. We start with the following notation:

– Let Γ be a randomized protocol which computes the optimal solution of the economic dispatch problem.

– At the beginning of Γ, each node i has 5 private inputs a_i, b_i, c_i, $\underline{x_i}$ and $\overline{x_i}$ without any probability space associated with them. We can represent private inputs of all nodes as Z, a $5 \times n$ matrix where i^{th} column is $\overrightarrow{z_i} = (a_i, b_i, c_i, \underline{x_i}, \overline{x_i})$ and 1^{st} row is $\overrightarrow{z_a} = (a_1, a_2, .., a_{n-1})$.

– $view^{\Gamma}_i$ is the set of information received by the i^{th} node during an execution of the protocol Γ.

Then protocol Γ under a non-colluding semi-honest adversary should satisfy the following privacy property:

Definition 8.5.1. *(Privacy)* None of the nodes should learn anything more than what follows from its input and allowed output from the execution of the protocol. A non-colluding semi-honest adversary i does not learn any additional information from the execution of Γ if the following holds:
For any two input matrices Z and W, which find the same output for the i^{th} generator node (i.e., if the protocol ends at round T, then $x_i(T) = output^{\Gamma}_i(Z) = output^{\Gamma}_i(W)$) and agree with i such that $\overrightarrow{z_i} = \overrightarrow{w_i}$, the probability distributions of the set of information received by node i are equal i.e. each information received does not depend on any input. Hence, $\Pr(view^{\Gamma}_i|Z) = \Pr(view^{\Gamma}_i|W)$

$output_i^\Gamma$ consists of $\sum_{\forall i \in V} \lambda_i(t)$ and $\sum_{\forall i \in V} y_i(t)$ for $t = \{0, 1, ..\}$. This security definition guarantees in the worst case that any adversary only gains knowledge of $output_i^\Gamma$. This is not the strongest security notion, where any adversarial node i would only gain knowledge of the optimal power output of node i. However, even with the knowledge of $output_i^\Gamma$ or $\sum_{\forall i \in V} \lambda_i(t)$ and $\sum_{\forall i \in V} y_i(t)$ for $t = \{0, 1, ..\}$, the adversarial node i can not gain any information about $(a_j, b_j, c_j, \overline{x_j}, \underline{x_j})$ or the optimal x_j for $j \neq i$. Note that even if the attacker node i knows D_j for some other node j, the attacker can only gain knowledge that $y_j(0)$ might be zero (if $\underline{x_j} \leq D_j \leq \overline{x_j}$ then $x_j(0) = D_j$) and that $\lambda_j(1)$ might be $\sum_{k \in V} \lambda_k(0)$ but everything else remains private as individual $\lambda_j(t)$ and $y_j(t)$ values are unknown.

Theorem 3. *(PPED protocol is private against any non-colluding semi-honest adversary i if $n \geq 4$.)*

Proof. The view of node j consists of $(\lambda_i^{(j)}(t), \lambda_{S_4}^i(t), y_i^{(j)}(t), y_{S_4}^i(t))$, for $i \in V, i \neq j, t \in \{0, 1, \ldots\}$. For node i, all of these are uniformly distributed with the constraints,

$$\sum_{i \in V} \lambda_i(t) = \sum_{i \in V, i \neq j} \lambda_i^{(j)}(t) + \sum_{i \in V, i \neq j} \lambda_{S_4}^i(t)$$
$$\sum_{i \in V} y_i(t) = \sum_{i \in V, i \neq j} y_i^{(j)}(t) + \sum_{i \in V, i \neq j} y_{S_4}^i(t)$$

This shows that the view of node i only depends on $output_i^\Gamma$, not on input matrices Z or W, unless $|V| = n \leq 3$ (in which case node j can recover values of $\lambda_i(t)$ for $i \neq j$). □

8.5.1 Communication Cost

Now, let us see how much communication overhead is produced by our PPED protocol compared to a non-private economic dispatch protocol. A non-private protocol finds the solution by sending $2tn(n-1)$ messages, whereas the PPED protocol takes $4tn(n-1)$. In PPED, for step 3 there are $n(n-1)$ messages sent for $\sum_{\forall i \in V} \lambda_i(t)$ and $n(n-1)$ messages sent for $\sum_{\forall i \in V} y(t)$. In step 5, another $2n(n-1)$ messages are sent. Hence, the total number of communication messages is $4tn(n-1)$ for a t round PPED protocol. In terms of order of complexity, both protocols have $\mathcal{O}(tn^2)$.

8.6 Communication Cost Improvement for PPED

In Yang *et al.* [8], at every round t, the IC $\lambda_i(t)$ and the mismatch $y_i(t)$ are broadcasted to other nodes. Every node needs to know $\sum_{i=1}^n \lambda_i(t)$ and $\sum_{i=1}^n y_i(t)$ at every round for ED calculation. We observe that each node could

still calculate $\sum_{i=1}^{n} \lambda_i(t)$ and $\sum_{i=1}^{n} y_i(t)$ with a reduced communication round than in Yang *et al.* [8] for a fully connected network. The communication rounds can be reduced by a factor of two for Yang *et al.* as follows:

1. Every node broadcasts only $\lambda_i(0)$ at $t = 0$.

2. Every node broadcasts $y_i(t)$ at every round.

We show how every node can calculate $\sum_{i=1}^{n} \lambda_i(t)$ and $y_i(t)$ at every round t from the aforementioned steps:

– At first at $t = 0$, each node finds: $\sum_{i=1}^{n} \lambda_i(0)$ and $\sum_{i=1}^{n} y_i(0)$.

– At $t = 1$, every node i finds:

$$\lambda_i(1) = \frac{1}{n} \sum_{i=1}^{n} \lambda_i(0) + \epsilon y_i(0)$$

Node i calculates power output:

$$x_i(1) = \beta_i \lambda_i(1) + \alpha_i$$

And the power mismatch at node i:

$$y_i(1) = \frac{1}{n} \sum_{i=1}^{n} y_i(t) - (x_i(1) - x_i(0))$$

Notice that:

$$\sum_{i=1}^{n} \lambda_i(1) = \sum_{i=1}^{n} \lambda_i(0) + \epsilon \sum_{i=1}^{n} y_i(0)$$

Hence, node i finds $\sum_{i=1}^{n} \lambda_i(1)$ as i knows $\sum_{i=1}^{n} \lambda_i(0)$, $\sum_{i=1}^{n} y_i(0)$, and the parameter ϵ.

– Therefore by mathematical induction, node i can calculate $\sum_{i=1}^{n} \lambda_i(t)$ and $y_i(t)$ at round t.

Similarity, the communication complexity of PPED can be reduced by a factor of two with this approach instead of the original Yang *et al.* protocol. Hence, the total number of communication messages will be $2tn(n-1)$ for t rounds.

8.7 Summary

We believe the attack scenario and our privacy solution will be applicable to other existing ED solutions. This work assumes a fully connected bidirectional topology. It would be interesting to extend this work to a relaxed topological constraint. Also, analyzing a stronger security notion with malicious adversaries is a future direction. Our assumption for the cost function here is quadratic as per the standard used by power engineers. However, the cost function could behave differently with different requirements in the smart grid. In the next chapter, we will analyze the security of ED solutions when the cost function changes to non-convex.

Chapter 9

Privacy-preserving ED Protocol II

9.1 Introduction

In this chapter, we provide our second privacy solution for distributed ED. The content of this chapter is previously published in [17] and [13]. We have shown a privacy-preserving distributed ED protocol for quadratic cost function in the previous chapter 8. In real-world smart grid scenarios, more practical considerations include valve point loading effects, multiple fuel options and prohibited operating zones. For example, to consider a valve-point loading effect, a sinusoidal term is added to the cost function with some non-differentiable points which makes the cost function non-convex. More details about the non-convex cost function for ED can be found in chapter 7, section 7.3. A realistic distributed ED solution is provided by Binetti *et al.* [9] for a non-smooth and non-convex cost function. In this chapter, we aim to solve the privacy problem in distributed ED with non-convex cost function. We use the same notations for ED problem defined in the Chapter 7, section 7.3. The main contributions of this work are as follows:

- We show that the ED protocol by Binetti *et al.* [9] leaks confidential information of the generator nodes.

- We transform the Binetti protocol into a privacy-preserving distributed protocol for ED calculation for smart grid systems.

- We analyze the security of our proposed protocol and give results of a prototype implementation of our protocol.

9.2 Binetti *et al.*'s ED Protocol [9]

9.2.1 System Model

We consider a power generation network consisting of a set $\mathcal{V}$ of m utility providers (UPs). Each UP P_i is a generator node with its own generation facility. The cost of power generation of a UP depends on its individual cost function. The UPs together are responsible for producing power for the consumers of a specific zone. The UPs generate power individually and feed it into the power line to meet the demand. We assume that the communication network between the UPs is time-synchronized and that each UP can send securely and anonymously, i.e., without revealing its ID, messages to any other UP. All power system measurements are treated as fixed-point numbers throughout our paper.

9.2.2 Protocol Description

All parties in $\mathcal{V}$ agree on a precision parameter s
All parties initialize power output to meet the demand
while *True* **do**
  Each party P_i sets $\pi_i(t)$ and $\mu_i(t)$:
  if $(\underline{x_i} \leq x_i(t) + s \leq \overline{x_i})$ **then**
    $\pi_i(t) := C_i(x_i(t) + s) - C_i(x_i(t)$
  else
    $\pi_i(t) := 0$
  if $(\underline{x_i} \leq x_i(t) - s \leq \overline{x_i})$ **then**
    $\mu_i(t) := C_i(x_i(t)) - C_i(x_i(t) - s)$
  else
    $\mu_i(t) := 0$
  Every P_i sends $\pi_i(t)$ (if $\neq 0$) and $\mu_i(t)$ (if $\neq 0$) to all other P_j)
  Every party $P_i \in \mathcal{V}$ finds $\bar{i}$ and $\bar{j}$ such that:
  $\pi_{\bar{i}}(t) = \min\{\pi_1(t), \ldots, \pi_m(t)\}$
  $\mu_{\bar{j}}(t) = \max\{\mu_1(t), \ldots, \mu_m(t)\}$
  Every party $P_i \in \mathcal{V}$ computes $\delta = \mu_{\bar{j}}(t) - \pi_{\bar{i}}(t)$
  if $(\delta > 0)$ **then**
    $x_{\bar{i}}(t+1) = x_{\bar{i}}(t) + s$
    $x_{\bar{i}}(t+1) = x_{\bar{i}}(t) + s$
  else
    Exit
  Increment t: $t := t + 1$
end

Algorithm 4: Binetti *et al.* Protocol

Binetti *et al.* [9] proposed an auction-based distributed consensus protocol.

It is a heuristic algorithm that can be shown analytically to get close to an optimal solution. Note that this is best solution, one can expect in practice as finding the global optimum in non-convex optimization is an NP-hard problem. The core idea of the Binetti *et al.* protocol is based on a *double auction* [137], where each UP can change their output power by negotiating with other UPs and drive the overall cost towards a global minimum.

Step 1: Bids Evaluation

In the initialization phase, all parties agree on a (non-optimal) output for each UP such that the global demand is met. Subsequently, they start a round-based protocol where with each round t, two UPs change their production by a fixed amount s. In order to determine the parties in question, each UP sends two bids $\pi_i(t)$ and $\mu_i(t)$ to all participants:

$$\pi_i(t) = C_i(x_i(t)+s) - C_i(x_i(t)) \tag{9.1}$$

$$\mu_i(t) = C_i(x_i(t)) - C_i(x_i(t)-s) \tag{9.2}$$

The $\pi_i(t)$ value denotes the estimated additional cost when increasing the power output from $x_i(t)$ to $x_i(t)+s$ while $\mu_i(t)$ is the estimated cost decrease when reducing the power production from $x_i(t)$ to $x_i(t)-s$. No bids are placed if the power increase or reduction violates the generator constraint equation (7.3).

Step 2: Consensus Procedure

The goal of the consensus process is to agree on the winning bids and the winners. First, each nodes sends its own bid $\pi_i(t)$ (bid π) and $\mu_i(t)$ (bid μ) to its neighbours. After receiving bids from the other nodes, every node updates the current value of the winning bid (as well as the bidder) as its local information and forwards. The node with the lowest value for $\pi_i(t)$ wins the bid π and the node with the highest value for $\mu_i(t)$ wins the bid μ. For example, each node achieves a consensus on the values $\pi_{\bar{i}}$,$\mu_{\bar{i}}$,$\bar{i}$ and $\bar{j}$:

$$\pi_{\bar{i}}(t) = \min_{i \in V} \pi_i(t)$$

$$\mu_{\bar{j}}(t) = \max_{i \in V} \mu_i(t)$$

Step 3: Auction Resolution and Swap Operation

The winner $\bar{i}$ is the node who can generate extra s units of power at the lowest additional cost. The winner $\bar{j}$ is the node who can save the most by reducing the power production by s units. If the difference $\delta = \mu_{\bar{j}}(t) - \pi_{\bar{i}}(t) > 0$, swapping the production of s units of power will lead to a cost reduction δ. Therefore, if $\delta > 0$, the update rule for $\bar{i}$ and $\bar{j}$ is:

$$x_{\bar{i}}(t+1) = x_{\bar{i}}(t) + s \quad \text{and} \quad x_{\bar{j}}(t+1) = x_{\bar{j}}(t) - s \tag{9.3}$$

The algorithm iterates until no exchange of s units of power between two nodes is possible to reduce the cost further. The demand constraint (Eq (7.2)) is always maintained as $D = \sum_{i=1}^{m} x_i(t)$ at any time t.

9.3 Security Model

In this section, we give the attacker model, privacy goals and show how the ED protocol by Binetti *et al.* leaks private information.

9.3.1 Attacker Model

We assume the grid infrastructure to be secure against outsider attackers, i.e., to be tamper-resistant and that no external malicious attacker can tamper with or insert false data without being detected. In a competitive energy market, the UPs could be malicious as well, e.g., may modify their input data to gain maximum profit or collude with other UPs to outplay their competitors. However, such behavior is risky since a convicted cheater might face a permanent ban from the energy market by the regulatory board. Consequently, we focus on the *honest-but-curious* adversary model. An honest-but-curious adversary will strictly follow the protocol specification but may analyze the messages exchanged during execution or collude with others to obtain private information about other participants. More precisely, in the considered scenario the aim is to derive information about the cost function and the upper and lower bounds on the power production. We assume an honest-but-curious internal attacker P_j that may be part of a colluding set $\mathcal{A} \subseteq \mathcal{V}$ of cardinality τ in the network.

9.3.2 Privacy Goals

The main attack motivation is that the attackers are interested in any specific UP's business information such as the cost function, e.g. in order to be able to choose a pricing strategy that will drive competitors out of the market. In [12], the author pointed out the privacy sensitivity of such information and demonstrated an attack against an existing distributed ED protocol for quadratic cost functions. If we consider non-convex cost functions, the non-convex parameters of the function should be also protected.

Thus, the privacy goal of our ED protocol is to protect the output power $x_i(t)$, the generator function parameters (e.g. a_i, b_i, c_i for a quadratic cost function), and the generator constraints $\underline{x_i}$ and $\overline{x_i}$ for every participant P_i.

9.3.3 Privacy Leakage in Binetti *et al.*'s Protocol

In the following, we demonstrate the need for a new privacy-preserving ED solution. To best of our knowledge, currently the most practicla real-world distributed ED solution is given by the protocol from Binetti *et al.* [9]. We show that it leaks the cost function parameters when an honest-but-curious

adversary P_j analyzes the information received during several iterations. The attacker P_j gets the value of $\pi_i(t)$ and $\mu_i(t)$ at $t = 0$:

$$\pi_i(0) = C_i(x_i(0) + s) - C_i(x_i(0)) = 2a_i x_i(0)s + a_i s^2 + b_i s \quad (9.4)$$

$$\mu_i(0) = C_i(x_i(0)) - C_i(x_i(0) - s) = 2a_i x_i(0)s - a_i s^2 + b_i s \quad (9.5)$$

Now, subtracting the equations (9.4) and (9.5) yields

$$2a_i s^2 = \pi_i(0) - \mu_i(0)) \implies a_i = \frac{\pi_i(0) - \mu_i(0)}{2s^2}.$$

Hence, a_i can be found as s is public and known to the attacker. Similarly, the value of b_i can be determined after a few rounds of iteration. In the case of a non-convex cost function, we can solve a system of non-linear equations constructed during several rounds to find the cost function parameters (e.g. with a numerical solver). Additionally, the initial power output is shared between the parties during the initialization which violates our privacy goals.

9.3.4 Cryptographic Building Blocks

Our solution makes use of several established cryptographic building blocks that we recap from chapter 2, section 2.3, in the following:

SMC: Secure Multiparty Computation (SMC) allows parties to compute a function over their input while their input values are kept private. Many different methods to perform SMC can be found in the existing literature [14, 15, 138, 28]. In particular, Ben-Or *et al.* [14] proposed a protocol (BGW protocol in the following) that allows to compute any function f with perfect security in the presence of τ honest-but-curious adversaries as long as $\tau < \frac{m}{2}$ (section 2.3, Thm. 1).

Shamir's Secret Sharing: The BGW protocol uses Shamir's secret sharing scheme [37]. In this algorithm, a secret can be divided into a number of unique shares such that a single share does not leak any information about the secret. In a (τ, m)-secret-sharing scheme, τ out of m shares are required to reconstruct the secret. Shamir's secret sharing scheme achieves information-theoretic security.

Secure Sum Protocol: A secure sum protocol calculates the addition function while keeping inputs of the parties private. A secure sum protocol can be constructed from some standard additive secret sharing scheme or by using some SMC (e.g. BGW) with addition gates. Some information-theoretic secure sum protocols can be found in [28, p. 8] [41].

9.4 Privacy-preserving Binetti (PPB) Protocol

The problem with the original Binetti protocol is that the $x_i(0)$ are shared between the parties for initialization, and cost function parameters are leaked in

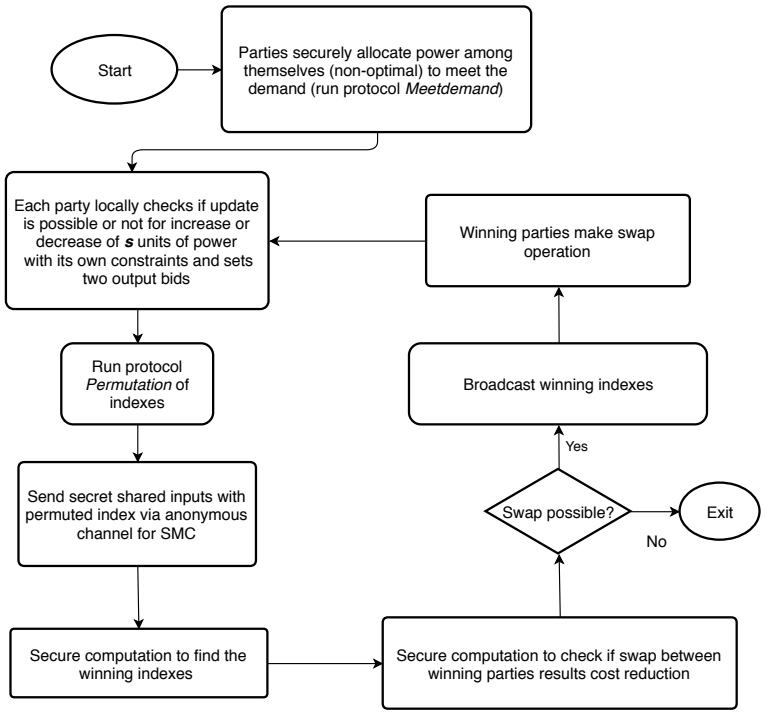

Figure 9.1: *PPB* Protocol Workflow

several iterations which violates our privacy goals. We thus introduce a privacy-preserving Binetti (*PPB*) protocol which preserves privacy of the individual UPs and correctly implements Binetti's protocol.

In this section, we give a high level overview of our *PPB* protocol and the intuition behind our construction. A straightforward solution would be to implement Binetti's protocol with SMC. However, here we face two challenges:

1. The Binetti protocol assumes that the UPs start with a configuration of the values x_i such that the demand is met. Consequently, we introduce a sub-protocol *Meetdemand* which at the beginning privately allocates the power output among m parties such that $\sum_{i=1}^{m} x_i(0) = D$. Note that this sub-protocol only allocates the power to meet the initial demand initially; optimization of the allocation is done in a second step.

2. In the Binetti protocol, each UP learns in each round the winning parties, i.e., the parties with the highest/lowest bids. To avoid this leakage, we introduce a sub-protocol for anonymous bid submission so only the winning parties will know if they are the winner. The sub-protocol *Permutation* securely permutes the indexes of the UPs such that a party will only know its own index but not those of the other parties.

Input: $\mathcal{X}_i$ as input for every party P_i
Output: $\mathcal{Y}_i$ for every party P_i such that total cost is minimum
Setup :
1 All parties $P_i \in \mathcal{V}$ agree on a prime field $\mathbb{F} = \mathbb{Z}_p$ for the SMC
Initialization:
2 All parties run protocol *Meetdemand()*:
$(P_1 : (D, m, \underline{x_1}, \overline{x_1}); \ldots; P_m : (D, m, \underline{x_m}, \overline{x_m})) \rightarrow (P_1 : x_1; \ldots; P_m : x_m)$
to get private x_i s.t. $\sum_{i=1}^{m} x_i = D$
3 Every P_i initialize $x_i(0) := x_i$
Main :
4 **while** *True* **do**
5 All parties run the protocol *Permutation()* to its permuted index:
$(P_1 : s_1; \ldots; P_m : s_m) \rightarrow (P_1 : ind(i); \ldots; P_m : ind(m))$
6 Every parties $P_i \in \mathcal{V}$ set $\pi_i(t)$ and $\mu_i(t)$ same as Binetti protocol
`/* (See section 9.2, algorithm 4, line. 4 to 12 */`
7 Each P_i creates Shamir's secret shares $[\pi_i(t)], [\mu_i(t)]$ of $\pi_i(t)$ and $\mu_i(t)$
8 Each party P_i sends shares $[\pi_i(t)]$ and $[\mu_i(t)]$ with its index $ind(i)$ to other $P_j \in \mathcal{V}$ $(i \neq j)$ through an anonymous secure channel for SMC
9 All parties $P_i \in \mathcal{V}$ run SMC to find $\overline{ind_\pi(i)}$ and $\overline{ind_\mu(i)}$ s.t.:
$\overline{ind_\pi(i)} := Index(\min(\pi_1(t), \ldots, \pi_m(t)))$
$\overline{ind_\mu(i)} := Index(\max(\mu_1(t), \ldots, \mu_m(t)))$
`/* Function` $Index()$ `returns the associated index` $ind(i)$ `if input is` $\pi_i(t)$ `or` $\mu_i(t)$ `*/`
10 All parties $P_i \in \mathcal{V}$ run SMC to evaluate a binary $Flag \in \{0, 1\}$ where $(Flag := (\delta > 0))$ and
$\delta = \max(\mu_1(t), \ldots, \mu_m(t)) - \min(\pi_1(t), \ldots, \pi_m(t))$
11 **if** $(Flag = 1)$ **then**
12 **if** $(ind(i) = \overline{ind_\pi(i)})$ **then**
13 $x_i(t+1) := x_i(t) + s$
14 **if** $(ind(i) = \overline{ind_\mu(i)})$ **then**
15 $x_i(t+1) := x_i(t) - s$
16 **else**
17 Exit
18 Increment t: $t := t + 1$
19 **end**

Algorithm 5: *PPB* Protocol

The workflow of our protocol is presented in Fig. 9.1. A more detailed overview follows. Here, we use the common notation for multi-party protocols. The expression

$$\Pi : (P_1 : \mathcal{X}_1; \ldots; P_m : \mathcal{X}_m) \rightarrow (P_1 : \mathcal{Y}_1; \ldots; P_m : \mathcal{Y}_m) \tag{9.6}$$

means that parties P_i commonly execute a protocol Π with with $\mathcal{X}_i$ and $\mathcal{Y}_i$ being their inputs and outputs, respectively.

At the beginning of *PPB*, the input $\mathcal{X}_i$ of each party P_i can be presented as $\mathcal{X}_i = (C_i, \underline{x_i}, \overline{x_i}, D, s, m)$. Here, the cost of function C_i, generator lower limit $\underline{x_i}$, and generator upper limit $\overline{x_i}$ are private whereas demand D, the precision parameter s and the number of participants m are known to every participant in the network. If t is the iteration index, then the private output is $\mathcal{Y}_i = x_i(t)$ such that the total cost $\sum_{i=1}^{m} C_i(x_i(t))$ is minimum. The *PPB* protocol is formally described in Algorithm 5. The general idea is as follows:

Step 1: Initialization At the start, all parties $P_i \in \mathcal{V}$ run the sub-protocol *Meetdemand* to get the initial value of power production $x_i(0)$. The protocol *Meetdemand* does not reveal the output $x_i(0)$ to other parties like in the original Binetti protocol. Each $x_i(0)$ meets the generator constraint of P_i and the total power production $\sum_{i=1}^{m} x_i(0)$ meets the current demand D. We describe protocol *Meetdemand* in detail in sub-section 9.4.1.

Step 2: Optimization Subsequently, the parties try to find the optimal solution $x_i(t)$ starting from $x_i(0)$ similarly to Binetti's protocol, but using SMC. If the concrete SMC protocol works with integer input, the fixed-point inputs can be converted by multiplying with a scaling factor. We also use Shamir secret sharing of the bids for the SMC input. The shared bids $\pi_i(t)$ and $\mu_i(t)$ are denoted by $[\pi_i(t)]$ and $[\mu_i(t)]$ respectively.

Step 3: Permutation As we explained above, a direct SMC realization of the auction will leak the winning indexes. So, instead of sending $\pi_i(t)$ and $\mu_i(t)$ bids directly for SMC, the parties send a permuted index along with the bids through an anonymous channel. The main auction function from Binetti is implemented with SMC (Algorithm 5, line 10-12) where it makes sure that the function input is kept private. The *Index()* function returns the associated permuted index $ind(i)$ if the input is $\pi_i(t)$ or $\mu_i(t)$. Henceforth, all parties after SMC will only find out the associated index of winning parties i.e. $\overline{ind_\pi(i)}$ and $\overline{ind_\mu(i)}$, however parties can not trace where the index actually came from (as it came through anonymous channel). Only the winning parties can determine whether they are the winner and will update accordingly like in Binetti's protocol. Protocol *Permutation* is explained in details in the sub-section 9.4.2.

It is obvious that *PPD* correctly implements the Binetti protocol if the protocols *Meetdemand* and *Permutation* are correct. In the following, we explain both protocols in detail and argue their correctness before we investigate the security in Section 9.5.

9.4.1 The *Meetdemand* Protocol

The sub-protocol *Meetdemand* is executed at the initial stage of our *PPB* protocol. It allows the UPs parties to allocate their respective power production x_i such that the demand D is met, i.e., $\sum_{i=1}^{m} x_i = D$. We assume that $\sum_{i=1}^{m} \underline{x_i} \leq D \leq \sum_{i=1}^{m} \overline{x_i}$, i.e. producing the current demand D is possible in principle. We describe in the following two different variants of *Meetdemand* that aim for different tradeoffs between efficiency and privacy. Variant 1 is more efficient as it allows to select appropriate values x_i immediately. The downside however is that each party gains information about the initial value $x_i(0)$. In contrast, variant 2 reveals no information about the values $x_i(0)$ but requires several iterations until a satisfying configuration has been found.

Variant I:

We denote by $\underline{x} := \sum_{i=1}^{m} \underline{x_i}$ and $\overline{x} := \sum_{i=1}^{m} \overline{x_i}$ the minimum and maximum amount of power that can be produced by all parties together. We assume that $\underline{x}$ and $\overline{x}$ are known to every party. One can use a secure sum protocol to get a $\underline{x}$ and $\overline{x}$ without revealing the private values for $\underline{x_i}$ and $\overline{x_i}$.

Furthermore, we define for each party P_i the distance between the upper and lower bound of power production as $d_i := \overline{x_i} - \underline{x_i}$. We denote the distance between lower and upper bound of total power production as $d := \overline{x} - \underline{x}$. Note that $d = \sum_{i=1}^{m} d_i$.

By assumption, it holds that $\underline{x} \leq D \leq \overline{x}$, i.e., the demand can be met by all parties. Thus, there exists an r such that $D = \underline{x} + r \cdot d$ with $0 \leq r \leq 1$. As D, $\underline{x}$ and $\overline{x}$ are known, the value of r is also know to every party. The strategy is that each party P_i contributes the same portion with respect to the power they can produce by setting its power production to $x_i = \underline{x_i} + r \cdot d_i$. Thus, the total initial production is

$$\sum_{i=1}^{m} x_i = \sum_{i=1}^{m} (\underline{x_i} + r \cdot d_i) = \sum_{i=1}^{m} \underline{x_i} + r \cdot (\sum_{i=1}^{m} d_i) = \underline{x} + r \cdot d = D,$$

i.e. the demand D is met exactly.

Variant II:

Variant I of the *Meetdemand* protocol is highly efficient but leaks the fraction r of $\overline{x_i} - \underline{x_i}$ (which in turn is secret) of each party P_i. Variant II (Algorithm 6) provides higher privacy but requires several iterations. First, each party P_i starts with a randomly chosen x_i within its range of its power production. Then, the participants determine $\sum_{i=1}^{m} x_i$ without revealing the individual x_i to each other by using a secure sum protocol. This $\sum_{i=1}^{m} x_i$ is also random as all x_i are. Then, every party can compute the amount of additional power needed to be allocated, i.e. $\Delta = \sum_{i=1}^{m} x_i - D$. Note that this Δ can be positive or negative. If Δ is positive (negative), the parties have to decrease (increase) the production.

Input: D, m, private $\underline{x_i}$ and $\overline{x_i}$ for every party $P_i \in \mathcal{V}$
Output: Private x_i for every party $P_i \in \mathcal{V}$ s.t. $\sum_{i=1}^{m} x_i = D$ and $\underline{x_i} \leq x_i \leq \overline{x_i}$
Meetdemand.Main:
Every party P_i sets a random x_i where $\underline{x_i} \leq x_i \leq \overline{x_i}$ and $k = m$
while *(*$|\Delta| > 0$*)* **do**
  All parties $P_i \in \mathcal{V}$ run a secure sum protocol to get $\sum_{i=1}^{m} x_i$
  Every party P_i finds: $\Delta := \sum_{i=1}^{m} x_i - D$
  At every party P_i:
  if *(*$\underline{x_i} \leq x_i - \frac{\Delta}{k} \leq \overline{x_i}$*)* **then**
    Set $x_i := x_i - \frac{\Delta}{k}$ and $Output_i := 1$
  if *(*$x_i - \frac{\Delta}{k} > \overline{x_i}$*)* **then**
    Set $x_i := \overline{x_i}$ and $Output_i := 0$
  if *(*$x_i - \frac{\Delta}{k} < \underline{x_i}$*)* **then**
    Set $x_i := \underline{x_i}$ and $Output_i := 0$
  All parties $P_i \in \mathcal{V}$ run a secure sum protocol to update:
  $k = \sum_{i=1}^{m} Output_i$
end

Algorithm 6: Variant II of the *Meetdemand* Protocol

The idea is that ideally all parties should change their power production by the same factor Δ/k where k denotes the number of parties than can still increase (resp. decrease) their power production. That is each party P_i checks if the new value $x_i := x_i - \frac{\Delta}{k}$ is within its production range and updates it accordingly if possible. Otherwise, P_i chooses the minimum ($\underline{x_i}$ for positive Δ) or maximum ($\overline{x_i}$ for negative Δ) limit and doesn't participate anymore in the following rounds. The number k of participating parties in the next round can be also found with a secure sum.

We quickly explain the correctness. Let Δ be the difference between $\sum_i x_i$ and D and k be the number of parties that can still adapt their power production. We show that there is at least one party that can adapt its power production so that the overall power production gets closer to the demand with each round. Without loss of generality, let $\Delta \geq 0$. Moreover, let I_0 be the indices of all parties that already produce the minimum power and I_1 the indexes of k parties that still participate.

Assume that it holds for all P_i with $i \in I_1$ that $x_i - \frac{\Delta}{k} < \underline{x_i}$, that is no party can further update its power production. Then it follows by definition of Δ that

$$D = \sum_{i=1}^{m} x_i - \Delta = \sum_{i \in I_0} x_i + \sum_{i \in I_1} x_i - \Delta = \sum_{i \in I_0} \underline{x_i} + \sum_{i \in I_1} (x_i - \frac{\Delta}{k}) < \sum_{i \in I_0} \underline{x_i} + \sum_{i \in I_1} \underline{x_i} = \underline{x}.$$

That is, the demand D would be outside of the range that can be produced, violating our initial assumption. Consequently, there needs to be at least one

party P_i that can further update the power production as long as $\Delta \neq 0$. This shows that eventually the demand will be met.

9.4.2 The *Permutation* Protocol

This sub-protocol allows to shuffle the indexes $ind(i)$ of the parties to allow to submit their bids anonymously during SMC. Formally, each of the m parties $\{P_1, \ldots, P_m\}$ gets one index $ind(i) \in \{1, \ldots, m\}$ such that

1. each P_i knows its index $ind(i)$ but does not know $ind(j)$ for $j \neq i$.
2. $\exists$ a bijective function $f : \{ind(1), \ldots, ind(m)\} \rightarrow \{1, \ldots, m\}$.

Permutation.Setup:
1 All parties agree on a cyclic group $(G, .)$ with generator g of order q, q being prime
2 Set $ind(i) := i$ for $i = 1, \ldots, m$
3 Each party P_i chooses a secret value $s_i \in \{2, \ldots, q-1\}$ and computes $h_i := g^{s_i}$
4 Each party P_i publishes its h_i to all parties
Permutation.Main($g, [h_1, \ldots, h_m]$):
5 **for** $i = 1$ **to** m **do**
6 | Party P_i takes the current parameter $((g, [h_1, \ldots, h_m])$
7 | P_i chooses a random value $r \in \{2, \ldots, q-1\}$ and a permutation $\Pi : \{1, \ldots, m\} \rightarrow \{1, \ldots, m\}$ to shuffle the indexes
8 | P_i updates: $(g, [h_1, \ldots, h_m]) \leftarrow (g^r, [{h_{\Pi[1]}}^r, \ldots, {h_{\Pi[m]}}^r]$
9 | Party P_i publishes $(g, [h_1, \ldots, h_m])$ to all parties and all parties take it as the current parameter
10 **end**
11 **for** $i = 1$ **to** m **do**
12 | Party P_i takes the current parameter $((g, [h_1, \ldots, h_m]$
13 | P_i computes $h^* := g^{s_i}$
14 | P_i finds index $j \in \{1, \ldots, m\}$ such that $h_j = h^*$
15 | P_i sets $ind(i) = j$
16 **end**

Algorithm 7: Protocol *Permutation*

The details of the protocol are given in Algorithm 7. It uses the Decisional Diffie-Hellman Problem (DDH) [139] to securely permute a sequence of indexes. Recall that the DDH is to distinguish tuples (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) from each other when a, b and c are chosen randomly and independently from $\mathbb{Z}_q$ and is considered to be hard.

First, all parties agree on a cyclic group $(G, .)$ with generator g of order q for a sufficiently large prime q. Every party P_i chooses a secret $s_i \in \{2, \ldots, q-1\}$ and computes $h_i := g^{s_i}$. The initial sequence of indexes is then specified by

$(g, [h_1, \ldots, h_m])$. Then, the parties round-wise permute the positions of the values h_i (via some randomly chosen permutation $\Pi : \{1, \ldots, m\} \to \{1, \ldots, m\}$) and update each h_i to $h_i^* := h_i^r$ and g to g^r for some random number $r \in \{2, \ldots, q-1\}$ as explained in lines $6-9$ of Algorithm 7. After each party applied the transformation, the final result $(g^*, [h_1^*, \ldots, h_m^*])$ is published and made available to all parties.

Then, each party P_i can determine its new location, i.e., index $ind(i)$, by finding the index such that $h^*_{ind(i)} = (g^*)^{s_i}$. Such a location exists as it holds that $\log_g(h_i) = \log_{g^r}(h_i^r)$ for each i. Moreover, if q is large, it holds with overwhelming probability that the values h_i are pairwise different. Due to the fact that the mapping $h \mapsto h^r$ is a permutation (as q is prime), it follows that the updated values h_i^r are pairwise different as well.

With respect to security, assume an adversary who aims to link two values from two successive rounds. That is, given $(g, [h_1, \ldots, h_m])$ and $(g^*, [h_1^*, \ldots, h_m^*])$, the adversary aims to decide for any two values h_i and h_j^* whether h_j^* is the updated value of h_i, i.e., if they are "linked". Note that this is equivalent to decide if (g^*, h_i, h_j^*) has the form $(g^r, g^{s_i}, g^{r \cdot s_i})$ for unknown values r and s_i, which means to solve an instance of the DDH. As this is assumed to be hard (for large values of q), it follows that an attacker cannot track the values h_i and in particular cannot determine the new index of other parties.

9.4.3 SMC Protocols

In every iteration t in *PPB*, the parties can run some SMC protocols (e.g., BGW [14] or SPDZ [138]) to evaluate the following functions:

1. Find the permuted index of the party who wins π auction.
2. Find the permuted index of the party who wins μ auction.
3. Find the value of $Flag \in \{0, 1\}$ which tells whether $\delta > 0$ or not.

9.5 Security Analysis of PPB Protocol

In the following, we discuss the information that is leaked by *PPB*. Recall that its main components are the protocols *Meetdemand*, *Permutation*, and the SMC protocols.

With respect to the latter, it is known that they reveal no information beyond what can be learned from the individual inputs and outputs. Likewise, we showed that *Permutation* leaks no information about the new index as long as the DDH instance is hard.

The only potential critical component is *Meetdemand* (where we focus on the more secure variant II). Here, any party learns the sequence by which it reaches the solution, that is for each round the value k, i.e. the number of participating parties in each iteration, and the $\sum_{i=1}^{m} x_i$ of values in each iteration. However, as the values x_i are randomly chosen at the beginning, we conjecture that this leakage is not harmful.

No. of UPs (m)	Average time (ms)
2	557
5	5484
10	11707
20	24186
50	34677
100	49986

Table 9.1: Performance of single round SMC execution with BGW

9.6 Implementation Observation

We have implemented a proof of concept prototype of our privacy-preserving protocol (single round). We used the BGW-based protocol suite from the Fresco Framework [140], a Java framework for secure computation. We consider 3 dedicated SMC nodes in our setting. Having a small number of computational nodes is recommended as the computational effort for SMC increases dramatically with increasing the number of nodes. As the security bound for BGW for τ honest-but-curious colluding adversary is $\tau < \frac{m}{2}$, we choose $\tau = 1$ for 3 SMC servers. We have performed our tests on a single 64 bit server with an Intel Core i5-4590 processor (4 cores) with 3.30 GHz and 8 GB RAM. We run our protocol locally for the 3 party setting with each instance run on a single core. Table 9.1 gives us the performance metrics for different numbers of UPs for a single round SMC execution as described in subsection 9.4.3.

We believe the results are reasonably good since in a realistic market the number m is small. Moreover, the frequency of ED calculation in current set-up ranges from hourly to once every few days. A report [141] by the Federal Energy Regulatory Commission (FERC) of California states that currently, economic dispatch is calculated hourly basis in a power grid. In the state of Baden Württemberg, Germany, there are 12 main distributed system operators (DSO) [142]. Now, if these 12 DSOs want to calculate ED privately on an hourly basis, our PPB protocol can run around 250-300 rounds using the prototype implementation and achieve a reasonably optimal solution. Also, we believe that performance results can be improved significantly by using distributed cloud servers with multiple cores, parallel processing of the data, and careful design of the algorithms.

9.7 Summary

We made the first step towards privacy-preserving solutions for the distributed economic dispatch problem. We hope that this works initiates further research into this area. In fact, there are several directions for improving of our protocol and its usability in a real world application.

For instance, the leakage of the *Meetdemand* protocol requires further inves-

tigation. While we do not expect any harmful leakage due to the fact that the starting values are randomly chosen, this does not exclude such possibility.

Another direction is to improve the implementation of the protocols, e.g., by using hybrid SMC protocols [138], by using some optimized technique for secure comparison functions [143], or by exploiting parallel processing of multiplications for bit-wise comparison.

Our distributed privacy solution ED is designed for a non-competitive market. The next step would be to design a privacy ED solution in a competitive market and against malicious attackers.

Chapter 10

Conclusions

In this thesis, we studied a few popular consensus-based protocols in blockchain and smart grid in the light of security and privacy.

We started with Part I, where we gave some preliminaries for this dissertation. First, we briefly overviewed the traditional consensus concepts and mentioned important results. Then we gave readers some mathematics and cryptography basis on graph theory and algebraic structures. Finally, in Part I, we describe a well-known cryptographic primitive secret sharing based secure multiparty computation (SMC).

In part II, we first discussed blockchain-related concepts. Then we briefly reviewed different blockchain consensus algorithms with the goal of byzantine tolerant. We saw that these blockchain consensus algorithms are often relaxed (e.g., *eventual synchrony*) than the traditional ones due to their practical specifications and criteria. Henceforth, new research area has emerged in the last few years as these algorithms are non-standard and solving its security problems brings new challenges. To the best of our knowledge, we made the first academic effort to analyze the security provisions of one of the most popular blockchain payment system, *Ripple*. In chapter 4, we describe the Ripple network, its functionality, and provided a survey of current Ripple related research results. In chapter 5, we discussed the Ripple's consensus protocol in detail and provided our security analysis. We showed that the Ripple consensus protocol parameters [5] do not prevent a fork in the system. We then provided the safety conditions for no fork in the system. Our findings led to further analysis of the protocol in [81], and further confirmed the centralized deployment of the protocol. We hope our findings will facilitate the need for rigorous analysis and improvement for the decentralization of Ripple consensus protocol.

Chapter 6 of part III introduced a basic overview of the smart grid, its agents and components, security and privacy problems, and different privacy-enhancing technologies. In chapter 7, we described the formulation of a fundamental optimization problem in smart grid known as *Economic Dispatch* (ED), and surveyed its security and privacy-related works. In chapter 8 and 9, we showed attacks against state of the arts distributed consensus-based ED solutions and

provided two privacy solutions based on Yang *et al.* (2013) and Binetti *et al.* (2014) with SMC primitives . Our both privacy solutions assumed a semi-honest attacker model and a fully connected topology. Future work includes the design of privacy-preserving ED protocols in a stronger attacker model, in relaxed network topology, and performance improvements.

Bibliography

[1] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.

[2] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[3] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[4] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, 1999.

[5] D. Schwartz, N. Youngs, and A. Britto. The Ripple protocol consensus algorithm. `https://ripple.com/files/ripple_consensus_whitepaper.pdf`, 2014. Accessed: 10-04-2019.

[6] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. `https://gavwood.com/Paper.pdf`, 2014. Accessed: 10-04-2019.

[7] Androulaki *et al.* Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 30:1–30:15. ACM, 2018.

[8] S. Yang, S. Tan, and J. X. Xu. Consensus based approach for economic dispatch problem in a smart grid. *IEEE Transactions on Power Systems*, 28(4):4416–4426, 2013.

[9] G. Binetti, A. Davoudi, D. Naso, B. Turchiano, and F. L. Lewis. A distributed auction-based algorithm for the nonconvex economic dispatch problem. *IEEE Transactions on Industrial Informatics*, 10(2):1124–1132, May 2014.

[10] Y. Cao, W. Yu, W. Ren, and G. Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Trans. Industrial Informatics*, 9(1):427–438, 2013.

[11] Ripple - one frictionless experience to send money globally using the power of blockchain. `https://www.ripple.com/`. Accessed: 30-08-2019.

[12] A. Mandal. Privacy preserving consensus-based economic dispatch in smart grid systems. In *International Conference on Future Network Systems and Security (FNSS 2016)*, volume CCIS 670, pages 98–110. Springer, 2016.

[13] A. Mandal, F. Armknecht, and E. Zenner. Privacy-preserving distributed economic dispatch protocol for smart grid. In *23rd Nordic Conference on Secure IT Systems (NordSec 2018)*, pages 3–18. Springer, 2018.

[14] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC 1988*, pages 1–10. ACM, 1988.

[15] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC 2006*, pages 285–304. Springer Berlin Heidelberg, 2006.

[16] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner. Ripple: Overview and outlook. In *8th International Conference on Trust and Trustworthy Computing (TRUST 2015)*, volume LNCS 9229, pages 163–180. Springer, 2015.

[17] A. Mandal and E. Zenner. Poster: Privacy in distributed economic dispatch in smart grid. In *2nd IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017. `https://www.ieee-security.org/TC/EuroSP2017/posters/poster13.pdf`. Accessed: 10-04-2019.

[18] A. Mandal, P. Wozniak, O. Vauderwange, and D. Curticapean. Application of visual cryptography for learning in optics and photonics. In G. Groot Gregory, editor, *Optics Education and Outreach IV*, volume 9946, pages 279 – 286. International Society for Optics and Photonics, SPIE, 2016.

[19] P. Wozniak, O. Vauderwange, A. Mandal, N. Javahiraly, and D. Curticapean. Possible applications of the LEAP motion controller for more interactive simulated experiments in augmented or virtual reality. In G. Groot Gregory, editor, *Optics Education and Outreach IV*, volume 9946, pages 234 – 245. International Society for Optics and Photonics, SPIE, 2016.

[20] D. Curticapean, O. Vauderwange, P. Wozniak, and A. Mandal. The International Year of Light 2015 and its impact on educational activities. In G. Groot Gregory, editor, *Optics Education and Outreach IV*, volume 9946, pages 80 – 88. International Society for Optics and Photonics, SPIE, 2016.

[21] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[22] R. Wattenhofer. *The Science of the Blockchain*. CreateSpace Independent Publishing Platform, USA, 1st edition, 2016.

[23] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[24] A. C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164. IEEE Computer Society, 1982.

[25] J. C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret (extended abstract). In *Advances in Cryptology – CRYPTO 1986*, pages 251–260. Springer Berlin Heidelberg, 1987.

[26] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, July 1991.

[27] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC 2006*, pages 285–304. Springer Berlin Heidelberg, 2006.

[28] R. Cramer, I. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[29] D. Beaver. Foundations of secure interactive computing. In *Annual International Cryptology Conference*, pages 377–391. Springer, 1991.

[30] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[31] O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[32] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):5, 2009.

[33] E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. *SIAM J. Comput.*, 39(5):2090–2112, March 2010.

[34] A. Aly. *Network flow problems with secure multiparty computation*. PhD thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2015. `https://dial.uclouvain.be/pr/boreal/object/boreal%3A165269/datastream/PDF_01/view`. Accessed: 10-04-2019.

[35] A. C. Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.

[36] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229. ACM, 1987.

[37] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[38] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, pages 313–317. AFIPS Press, 1979.

[39] A. Beimel. Secret-sharing schemes: A survey. In *Proceedings of the Third International Conference on Coding and Cryptology*, IWCC'11, pages 11–46. Springer-Verlag, 2011.

[40] G. Asharov and Y. Lindell. A full proof of the bgw protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, 2017.

[41] J. C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Advances in Cryptology, Crypto 86*, pages 251–260. Springer, 1986.

[42] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *PODC*, volume 98, pages 101–111. ACM, 1998.

[43] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `http://bitcoin.org/bitcoin.pdf`, 2009. Accessed: 30-04-2019.

[44] C. Cachin and M. Vukolic. Blockchain consensus protocols in the wild. *CoRR*, abs/1707.01873, 2017.

[45] D. Yaga, P. Mell, N. Roby, and K. Scarfone. NISTIR 8202: Blockchain technology overview. `https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf`, 2018. Accessed: 30-08-2019.

[46] ISO/TC 307. Blockchain and distributed ledger technologies. `https://www.iso.org/committee/6266604.html`, 2017. Accessed: 30-08-2019.

[47] G. O. Karame. On the security and scalability of Bitcoin's blockchain. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1861–1862. ACM, 2016.

[48] S. Bano, A. Sonnino, M. A. Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis. Consensus in the age of blockchains. *CoRR*, abs/1711.03936, 2017.

[49] J. A. Garay, A. Kiayias, and N. Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, pages 281–310. Springer, 2015.

[50] C. Decker. *On the scalability and security of Bitcoin.* PhD thesis, ETH Zurich, 2016. `https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/114732/eth-48881-02.pdf`. Accessed: 10-04-2019.

[51] A. N. Bessani, J. Sousa, and E. A. P. Alchieri. State machine replication for the masses with BFT-SMART. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*, pages 355–362, 2014.

[52] E. Buchman, J. Kwon, and Z. Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.

[53] K. Joyes. Safety, liveness and fault tolerance - the consensus choices. `https://www.stellar.org/blog/safety_liveness_and_fault_tolerance_consensus_choice/`, 2014. Accessed: 30-08-2019.

[54] C. Cachin and B. Tackmann. Asymmetric distributed trust. *CoRR*, abs/1906.09314, 2019.

[55] D. Mazieres. The Stellar consensus protocol: A federated model for internet-level consensus. `https://www.stellar.org/papers/stellar-consensus-protocol.pdf`, 2015. Accessed: 30-08-2019.

[56] M. Kim, Y. Kwon, and Y. Kim. Is Stellar as secure as you think? *CoRR*, abs/1904.13302, 2019.

[57] D. Malkhi and M. K. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

[58] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68, 2017.

[59] V. Buterin and V. Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.

[60] R. Fugger. Money as IOUs in social trust networks a proposal for a decentralized currency network protocol. `http://library.uniteddiversity.coop/Money_and_Economics/decentralizedcurrency.pdf`, 2004. Accessed: 30-08-2019.

[61] Ripplepay: A Ripple communication company. `https://classic.ripplepay.com/`. Accessed: 30-08-2019.

[62] Ripple Insights. Ripplenet surpasses 200 customers worldwide. https://www.ripple.com/insights/ripplenet-surpasses-200-customers-worldwide/, Accessed: 30-08-2019.

[63] A. Liu. Santander: Distributed ledger tech could save banks $20 billion a year. https://www.ripple.com/insights/santander-distributed-ledger-tech-could-save-banks-20-billion-a-year/. Accessed: 30-08-2019.

[64] Top 100 cryptocurrencies by market capitalization. https://coinmarketcap.com. Accessed: 30-08-2019.

[65] An explanation of Ripple's XRP escrow. https://xrpl.org/blog/2017/explanation-of-ripples-xrp-escrow.html. Accessed: 30-08-2019.

[66] XRP charts. https://coinmarketcap.com/currencies/ripple/#charts. Accessed: 30-08-2019.

[67] Ripple Labs Inc. XRP Charts. https://xrpcharts.ripple.com. Accessed: 08-09-2019.

[68] XRP Stats. Richlist stats. https://ledger.exposed/rich-stats. Accessed: 08-09-2019.

[69] A. Ghosh, M. Mahdian, D. M. Reeves, D. M. Pennock, and R. Fugger. Mechanism design on trust networks. In *Proceedings of the 3rd International Conference on Internet and Network Economics*, WINE'07, pages 257–268. Springer-Verlag, 2007.

[70] International Ripple Business Association. Ripple market makers. http://www.xrpga.org/market-makers.html. Accessed: 04-09-2014.

[71] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina. Privacy preserving payments in credit networks: Enabling trust with privacy in online marketplaces. In *Network and Distributed System Security (NDSS) Symposium*, 2015.

[72] Ripple Labs Inc. XRP ledger: Transaction formats. https://xrpl.org/transaction-formats.html. Accessed: 30-08-2019.

[73] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. In *24th Annual Network and Distributed System Security Symposium, NDSS*, 2017.

[74] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. In *25th Annual Network and Distributed System Security Symposium, NDSS*, 2018.

[75] A. Di Luzio, A. Mei, and J. Stefa. Consensus robustness and transaction de-anonymization in the Ripple currency exchange system. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 140–150, June 2017.

[76] P. Moreno-Sanchez. *Credit Network Payment Systems: Security, Privacy and Decentralization*. PhD thesis, Purdue University, 2018. `https://docs.lib.purdue.edu/dissertations/AAI10830436/`, Accessed: 10-04-2019.

[77] P. Moreno-Sanchez, N. Modi, R. Songhela, A. Kate, and S. Fahmy. Mind your credit: Assessing the health of the Ripple credit network. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 329–338, 2018.

[78] P. Moreno-Sanchez, M. B. Zafar, and A. Kate. Listening to whispers of Ripple: Linking wallets and deanonymizing transactions in the Ripple network. *PoPETs*, 2016(4):436–453, 2016.

[79] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, Oct 2001.

[80] P. Moreno-Sanchez, T. Ruffing, and A. Kate. Pathshuffle: Credit mixing and anonymous payments for Ripple. *PoPETs*, 2017(3):110, 2017.

[81] B. Chase and E. MacBrough. Analysis of the XRP ledger consensus protocol. *CoRR*, abs/1802.07242, 2018. `http://arxiv.org/abs/1802.07242`. Accessed: 08-03-2019.

[82] Ripple Labs Inc. Wiki. `https://wiki.ripple.com`. Accessed: 27-11-2014.

[83] G. O. Karame, E. Androulaki, and S. Capkun. Double-spending fast payments in Bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 906–917. ACM, 2012.

[84] V. Buterin. Bitcoin network shaken by blockchain fork. `https://bitcoinmagazine.com/3668/bitcoin-network-shaken-by-blockchain-fork/`, 2013. Accessed: 10-04-2019.

[85] Ripple Labs. Correction to Ripple white paper. `https://ripple.com/insights/correction-to-ripple-white-paper/`, 2015. Accessed: 03-04-2019.

[86] H. Farhangi. The path of the smart grid. *IEEE Power and Energy Magazine*, 8(1):18–28, January 2010.

[87] Y. Yan, Y. Qian, H. Sharif, and D. Tipper. A survey on smart grid communication infrastructures: Motivations, requirements and challenges. *IEEE Communications Surveys Tutorials*, 15(1):5–20, 2013.

[88] G. Danezis M. Jawurek, F. Kerschbaum. Sok: Privacy technologies for smart grids - a survey of options. *Microsoft Res., Cambridge, UK*, 2012.

[89] European Commission. Vision and strategy for Europe's electricity networks of the future. `https://ec.europa.eu/research/energy/pdf/smartgrids_en.pdf`. Accessed: 10-04-2019.

[90] European Commission. Standardization mandate to European standardisation organisations (ESOs) to support european smart grid deployment. `https://ec.europa.eu/energy/sites/ener/files/documents/2011_03_01_mandate_m490_en.pdf`. Accessed: 10-04-2019.

[91] M. Jawurek. *Privacy in Smart Grids*. PhD thesis, University of Erlangen-Nuremberg, 2013. `http://opus4.kobv.de/opus4-fau/frontdoor/index/index/docId/3645`. Accessed: 10-04-2019.

[92] European Commission. Impact assessment support study on "policies for DSOs, distribution tariffs and data handling". `https://ec.europa.eu/energy/sites/ener/files/documents/ce_vva_dso_final_report_vf.pdf`. Accessed: 10-04-2019.

[93] R. R. Mohassel, A. Fung, F. Mohammadi, and K. Raahemifar. A survey on advanced metering infrastructure. *Electrical Power and Energy Systems*, 63:473484, 2014.

[94] European Parliament & Council. European parliament & council. directive 2009/72/ec of the European parliament and of the council concerning common rules for the internal market in electricity and repealing directive 2003/54/ec. *Official Journal of the European Union*, L 211:5593, 2009.

[95] B. H. Chowdhury and S. Rahman. A review of recent advances in economic dispatch. *IEEE Transactions on Power Systems*, 5(4):1248–1259, 1990.

[96] S. Cleemput. *Secure and privacy-friendly smart electricity metering*. PhD thesis, KU LEUVEN, 2018. `https://www.esat.kuleuven.be/cosic/publications/thesis-303.pdf`. Accessed: 10-04-2019.

[97] V. C. Gungor, D. Sahin, T. Koçak, S. Ergüt, C. Buccella, C. Cecati, and G. P. Hancke. Smart grid technologies: Communication technologies and standards. *IEEE Trans. Industrial Informatics*, 7(4):529–539, 2011.

[98] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke. A survey on smart grid potential applications and communication requirements. *IEEE Transactions on Industrial Informatics*, 9(1):28–42, Feb 2013.

[99] P. McDaniel and S. McLaughlin. Security and privacy challenges in the smart grid. *IEEE Security Privacy*, 7(3):75–77, May 2009.

[100] C. Zhao, J. He, P. Cheng, and J. Chen. Analysis of consensus-based distributed economic dispatch under stealthy attacks. *IEEE Transactions on Industrial Electronics*, 64(6):5107–5117, June 2017.

[101] Y. Wang, D. Ruan, D. Gu, J. Gao, D. Liu, J. Xu, F. Chen, F. Dai, and J. Yang. Analysis of smart grid security standards. In *2011 IEEE International Conference on Computer Science and Automation Engineering*, volume 4, pages 697–701, June 2011.

[102] OSGP. Technical information. `http://www.osgp.org/en/technical`. Accessed: 10-04-2019.

[103] P. Jovanovic and S. Neves. Practical cryptanalysis of the open smart grid protocol. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, pages 297–316, 2015.

[104] Y. Yan, Y. Qian, H. Sharif, and D. Tipper. A survey on cyber security for smart grid communications. *IEEE Communications Surveys and Tutorials*, 14(4):998–1010, 2012.

[105] E-ISAC. Analysis of the cyber attack on the ukrainian power grid. `https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf`, 2016. Accessed: 10-04-2019.

[106] S. D. Warren and L. D. Brandeis. The Right to Privacy. *Harvard Law Review*, page 4(5):193220, 1890.

[107] A. Westin. Privacy and Freedom. *Bodley Head*, 1970.

[108] A. Abidin, A. Aly, S. Cleemput, and M. A. Mustafa. An MPC-based privacy-preserving protocol for a local electricity trading market. In *Cryptology and Network Security - 15th International Conference, CANS 2016*, pages 615–625, 2016.

[109] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management, 2009. `http://dud.inf.tu-dresden.de/Anon_Terminology.shtml`. Accessed: 10-04-2019.

[110] Z. Huang, S. Mitra, and N. Vaidya. Differentially private distributed optimization. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking, ICDCN 2015*, pages 4:1–4:10, 2015.

[111] L. Wu and J. Li. Privacy-preserving economic dispatch in competitive electricity market. In *2018 IEEE/PES Transmission and Distribution Conference and Exposition (T&D)*, pages 1–5, April 2018.

[112] F. Li, B. Luo, and P. Liu. Secure information aggregation for smart grids using homomorphic encryption. In *2010 First IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 327–332. IEEE, 2010.

[113] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 175–191. Springer, 2011.

[114] Z. Erkin, J. R. Troncoso-Pastoriza, R. L. Lagendijk, and F. Perez-Gonzalez. Privacy-preserving data aggregation in smart metering systems: An overview. *IEEE Signal Processing Magazine*, 30(2):75–86, 2013.

[115] H. Li, X. Lin, H. Yang, X. Liang, R. Lu, and X. Shen. EPPDR: an efficient privacy-preserving demand response scheme with adaptive key evolution in smart grid. *IEEE Trans. Parallel Distrib. Syst.*, 25(8):2053–2064, 2014.

[116] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In *HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy*, page 10, 2013.

[117] J. T. Mühlberg, S. Cleemput, M. A. Mustafa, J. Van Bulck, B. Preneel, and F. Piessens. An implementation of a high assurance smart meter using protected module architectures. In *IFIP International Conference on Information Security Theory and Practice*, pages 53–69. Springer, 2016.

[118] A. Bakirtzis, V. Petridis, and S. Kazarlis. Genetic algorithm solution to the economic dispatch problem. *IEE Proceedings-Generation, Transmission and Distribution*, 141(4):377–382, 1994.

[119] Z. L. Gaing. Particle swarm optimization to solving the economic dispatch considering the generator constraints. *IEEE Transactions on Power Systems*, 18(3):1187–1195, 2003.

[120] Z. Zhang and M. Y. Chow. Convergence analysis of the incremental cost consensus algorithm under different communication network topologies in a smart grid. *IEEE Transactions on Power Systems*, 27(4):1761–1768, 2012.

[121] A. J. Wood and B. F. Wollenberg. *Power Generation, Operation, and Control*. A Wiley-Interscience publication. Wiley, 1996.

[122] A. D. Dominguez-Garcia, S. T. Cady, and C. N. Hadjicostis. Decentralized optimal dispatch of distributed energy resources. In *2012 IEEE 51st Annual Conference on Decision and Control (CDC)*, pages 3688–3693. IEEE, 2012.

[123] D. Shelar, P. Sun, S. Amin, and S. A. Zonouz. Compromising security of economic dispatch in power system operations. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017*, pages 531–542, 2017.

[124] A. R. Borden, D. K. Molzahn, P. Ramanathan, and B. C. Lesieutre. Confidentiality-preserving optimal power flow for cloud computing. In *50th Annual Allerton Conference on Communication, Control, and Computing*, pages 1300–1307, 2012.

[125] C. Rottondi and G. Verticale. Privacy-friendly appliance load scheduling in smart grids. In *IEEE Fourth International Conference on Smart Grid Communications, SmartGridComm 2013*, pages 420–425, 2013.

[126] S. Chatzivasileiadis. DTU course 31765 lecture notes: Optimization in modern power systems. `https://arxiv.org/pdf/1811.00943.pdf`, 2018. Accessed: 10-04-2019.

[127] A. R. Borden, D. K. Molzahn, B. C. Lesieutre, and P. Ramanathan. Power system structure and confidentiality preserving transformation of optimal power flow problem. In *51st Annual Allerton Conference on Communication, Control, and Computing*, pages 1021–1028, 2013.

[128] D. Wu, B. C. Lesieutre, P. Ramanathan, and B. Kakunoori. Preserving privacy of AC optimal power flow models in multi-party electric grids. *IEEE Trans. Smart Grid*, 7(4):2050–2060, 2016.

[129] C. Zhao, J. He, P. Cheng, and J. Chen. Privacy-preserving consensus-based energy management in smart grid. In *2017 IEEE Power Energy Society General Meeting*, pages 1–5, July 2017.

[130] Z. Yang, P. Cheng, and J. Chen. Differential-privacy preserving optimal power flow in smart grid. *IET Generation, Transmission & Distribution*, 11(15):3853–3861, 2017.

[131] Y. Zhao, J. Yu, M. Ban, Y. Liu, and Z. Li. Privacy-preserving economic dispatch for an active distribution network with multiple networked microgrids. *IEEE Access*, 6:38802–38819, 2018.

[132] E. Liu and P. Cheng. Mitigating cyber privacy leakage for distributed dc optimal power flow in smart grid with radial topology. *IEEE Access*, 6:7911–7920, 2018.

[133] M. R. Sarker, J. Wang, Z. Li, and K. Ren. Security and cloud outsourcing framework for economic dispatch. *IEEE Trans. Smart Grid*, 9(6):5810–5819, 2018.

[134] Y. Lu and M. Zhu. Privacy preserving distributed optimization using homomorphic encryption. *Automatica*, 96:314–325, 2018.

[135] L. Wu. A transformation-based multi-area dynamic economic dispatch approach for preserving information privacy of individual areas. *IEEE Trans. Smart Grid*, 10(1):722–731, 2019.

[136] B. Chor and E. Kushilevitz. A communication-privacy tradeoff for modular addition. *Information Processing Letters*, 45(4):205–210, 1993.

[137] R. P. McAfee. A dominant strategy double auction. *Journal of Economic Theory*, 56(2):434 – 450, 1992.

[138] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, pages 643–662. Springer Berlin Heidelberg, 2012.

[139] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006.

[140] A framework for efficient secure computation. `http://fresco.readthedocs.io/en/latest/`. Accessed: 17-08-2017.

[141] FERC Staff. Economic Dispatch: Concepts, practices and issues. `https://www.ferc.gov/CalendarFiles/20051110172953-FERC%20Staff%20Presentation.pdf`. Accessed: 30-07-2018.

[142] Distribution network operators in Baden Württemberg. `https://www.energieatlas-bw.de/netze/verteilnetzbetreiber-strom`. Accessed: 30-07-2018.

[143] H. Lipmaa and T. Toft. Secure equality and greater-than tests with sublinear online complexity. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part II*, ICALP'13, pages 645–656, 2013.