5-1-2020

Authoritative and Unbiased Responses to Geographic Queries

Naresh Adhikari

Authoritative and unbiased geographic services

By

Naresh Adhikari

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2020

Authoritative and unbiased geographic services

By

Naresh Adhikari

Approved:

Mahalingam Ramkumar
(Major Professor)

Eric Hansen
(Committee Member)

Maxwell Young
(Committee Member)

Tanmay Bhowmik
(Committee Member)

T. J. Jankun-Kelly
(Graduate Coordinator)

Jason M. Keith
Dean
Bagley College of Engineering

Name: Naresh Adhikari

Date of Degree: May 1, 2020

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Mahalingam Ramkumar

Title of Study: Authoritative and unbiased geographic services

Pages of Study: 170

Candidate for Degree of Doctor of Philosophy

Trust in information systems stem from two key properties of responses to queries regarding the state of the system, viz., i) authoritativeness, and ii) unbiasedness. That the response is authoritative implies that i) the provider (source) of the response, and ii) the chain of delegations through which the provider obtained the authority to respond, can be verified. The property of unbiasedness implies that no system data relevant to the query is deliberately or accidentally suppressed. The need for guaranteeing these two important properties stem from the impracticality for the verifier to exhaustively verify the correctness of every system process, and the integrity of the platform on which system processes are executed. For instance, the integrity of a process may be jeopardized by i) bugs (attacks) in computing hardware like Random Access Memory (RAM), input/output channels (I/O), and Central Processing Unit(CPU), ii) exploitable defects in an operating system, iii) logical bugs

in program implementation, and iv) a wide range of other embedded malfunctions, among others.

A first step in ensuing AU properties of geographic queries is the need to ensure AU responses to a specific type of geographic query, viz., point-location. The focus of this dissertation is on strategies to leverage assured point-location, for i) ensuring authoritativeness and unbiasedness (AU) of responses to a wide range of geographic queries; and ii) useful applications like Secure Queryable Dynamic Maps (SQDM) and trustworthy redistricting protocol. The specific strategies used for guaranteeing AU properties of geographic services include i) use of novel Merkle-hash tree- based data structures, and ii) blockchain networks to guarantee the integrity of the processes.

Keywords: point-location, cryptography, security, regional delegation, block-chain, trusted-computing-base, TCB, congressional redistricting, SQDM

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

> We can only see a short distance ahead, but we can see plenty there that needs to be done..
>
> *Alan Turing, 1950*

This chapter introduces the problem of *point location*, and the motivation for the proposed research for securing geospatial data and the services based on them. We define and solve what is known as Assured Point Location Problem (APL) to achieve security assurances desired for geospatial data and services.

1.1 Motivation

A planar point location problem is a classical problem in combinatorial computational geometry [17]. As shown in Figure 1.1, let us consider a planar region R with n vertices and k sub-partitions. The polygonal partitions R_i and R_j are created by drawing a series of straight lines forming the boundary of the partitions. For a point location problem, we report the partition R that contains a given query point $q(x, y)$. If q lies on a vertex, or on the edge of R, then that vertex or edge is reported.

Based on the application, a Point Location Problem (PLP) can be of two broad types: 1) Static (Off-Line), and 2) Dynamic (On-Line). Static PLP permits *no*

modification to the polygonal structure. Dynamic PLP allows changes like addition/deletion of edges.

PLP has numerous applications in computer graphics, path planning in robotics, Geographic Information System (GIS), computer network management, map and navigation systems, database queries, data classification, and among others.

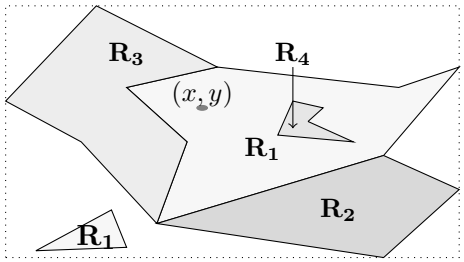

Figure 1.1

Given a query point $q(x, y)$, a point location algorithm reports region R_1.

Over thirty years, several variants of point location algorithms have been purposed and analyzed towards optimizing time and space requirements [5, 17, 30, 29, 51, 16, 50, 48, 26, 8, 40, 15]. However, a significant limitation of the existing algorithms lies in difficulty in guaranteeing the integrity of their execution. This is the motivation behind proposing Assured Point Location (APL) algorithms and protocols.

1.2 Research Topics

The broader scope of this research is securing spatial data assets, guaranteeing correctness of protocols for far-reaching applications and services that use geo-spatial data as chief commodities, and ensuring integrity of execution of such protocols. A

broad approach, '*transaction-based state machine*' [70], is adopted to secure such applications/services. The method will be elaborated in section 2.5.

1.2.1 Assured Point Location

Assured Point Location redefines a classical point location problem from the perspective of the security of an information system. Unlike classical PLP algorithms (see Section 2.2), APL provides trustworthy means to ascertain two critically essential properties of a process output, viz. Authoritative (A), and Unbiasedness (U).

1.2.1.1 Property of Authoritativeness

The property of authoritative ensures that the source of a response and the chain of trust through which the source obtained an authority of responding queries can be well-established. For instance, consider an Internet domain-name resolving service such as a Domain Name System (DNS). The top-level operator, also called root (/) of a DNS zone hierarchy can yield an authority of resolving NS queries to different Top Level Domain (TLD) zones such as *com, org, net, gov*, etc; TLD operators can themselves delegate authorities to different sub-domain zones. In doing so, an upper-level zone authority must sign the public key of the lower-level zone authorities, thus creating a chain of assigned trust. For instance, '*com*' zone authority signs a public key of Verisign to yield ownership of all names ending in '.com'. Verisign controls all the namespaces under the '.com' zone. It can sign the public key of Google Inc. to yield all names ending with 'google.com' to create a lower-level zone called 'google'. In this fashion, an authority transfer vertically to the lowest level possible. It is easy

to establish the chain of delegation for any authority up to the root (/) in a DNS hierarchy. DNSSSEC [36] protocol allows to authoritative delegation of namespaces; and also provides authoritative denial of services.

1.2.1.2 Property of Unbiasedness

The property of unbiasedness ensures that no valid answer instances have been deliberately or accidentally omitted/suppressed. It means that neither a malicious responder nor an attacker should be able to partially or entirely hide correct instances of a solution set, without being detected. For instance, a database SELECT query must return all the rows that satisfy a given search criteria. A map service, for example, must return all instances for a query such as "*n gas-stations near any point* (x, y)" must return complete answer instances; without hiding or suppressing any information. Similarly, a point location query must produce unbiased responses, otherwise, it should provide an ability to detect as such.

Having defined an APL, it is vital to understand the context of execution of classical algorithms – details of which are discussed in Section 2.2.

1.2.2 Attack/Threat Model(s)

A process in a digital (computer) system is a program in execution on a general-purpose computing platform. Its major components are i) a central processing unit (CPU), ii) input and outputs (I/O) channels, and iv) a temporary memory called Random Access Memory (RAM) [65]. Naturally, the reliability of a process output depends on 1) a level of isolation of the computing platform, and 2) the correctness

of the implemented programs. These are points where the reliability of a program output can be jeopardized. Malicious tampering of a computing platform (tampering CPU, RAM, CPU or I/O) can adversely impact program output, in the same way, it does when introducing deliberate (or inadvertent) program bugs [44, 60], [59], [38], [49], [69], [66], [60]. Given the fact that an application runs on top of a multitude of system programs, possibly having millions of lines of codes, it is virtually impossible to verify the correctness of such a complex operating system. Nevertheless, system developers and end-users have no choice but put trust in the integrity of an existing computing environment.

It safely leads to conclude that several lines of attacks that can lead a program to produce an incorrect solution for a point location problem too. A real-world point location program feeds on polygons, possibly with thousands of sides, enclosing planar regions. An attack on the integrity of the stored map data could adversely affect the reliability of a program output. The malicious or accidental bugs in the program implementation can produce subtly incorrect/incomplete output [44]. The potential forgeries by a point location process raise an acute need for guaranteeing authoritative and unbiased location query response. It can be served by *a trustworthy system that embodies the ability to attest the source of response to point location queries and establish the chain of trust (authorities) that leads to the source.* The goal of the proposed research is to design a transaction-based state-machine model that enables users to evaluate the integrity of the point location process and services that depend on the process.

Naresh et al. [1] proposed to secure queryable dynamic maps (SQDM) as a passive security model for a point location problem. The model is based on system-specific, rule-based certified state transitions that guarantees a specific relationship between input and output parameters. The model is an integration of a simple tamper-proof hardware that serves as a Trusted Computing Base (TCB), authenticated data structures (ADS), and a set of procedures exposed to facilitate and certify point location queries. It exposes limited operations that can be safely executed inside the TCB; it is unaffected by tampering of code/data that may reside outside the TCB. However, a significant limitation originates from the impracticality of the wide adoption of trusted hardware [6]. It also demands stringent design procedures to keep it highly tamper-proof. The limitations, nevertheless, can be addressed by translating the idea of the minimality of software and hardware to a system of peer-to-peer network sharing a universally verifiable distributed ledger – a blockchain network[12, 46, 70, 47]. Subtle differences between execution of APL protocols and GIS services on hardware-based TCB versus blockchain networks will be introduced in later chapters.

1.2.3 Applications of Assured Point Location

A conventional PL algorithm has utility in numerous applications such as computer graphics, database queries, autonomous vehicle tracking, and robotics, among others. There are other applications which call for a scalable and distributed secure point location protocols. For instance, secure biometrics, data classification, navi-

gation and positioning systems, digital watermarking, and copyright management, to name a few [68]. Among others, its applications in services that leverage spatial data are also in demand. We refer to them as *secure GIS services,* which are subsystems that use spatial data as useful commodities. Solutions to a secure (assured) point location problem open avenues to a wide range of secure geographic services. For instance, their utility can be extended to unmanned aircraft system navigation; GIS-based terrestrial navigation; autonomous vehicle tracking; real-estate transaction services without trusted third parties (like government agencies); authoritative, reliable, and unbiased geography-based information look-up services, etc.

1.2.3.1 Authoritative and Unbiased Geographic Delegation

Akin to transferring money from one's bank account to another or like transferring balance using cryptocurrency like Bitcoin on the Internet, delegating authority of geographic area without the involvement of any third party is an essential step towards fully automating transactions of land or property through the web. However, this seemingly trivial process has the following challenges to overcome.

1. Securing underlying geo-spatial data for region boundaries, feature locations, etc.
2. Ensuring an authority has the rightful authority over a 2D space.
3. Enforcing utmost trust in partition of a 2D region.
4. Secure and trustworthy transfer of authority over a sub-division to another authority.

A secure, reliable, and trustworthy system of delegating geographic region must address above mentioned challenges, as well as it must fulfill following prerequisites:

1. An authority holds proof of authority over a 2D space.
2. Such service must circumvent a sophisticated distributed denial of service (DDoS) and must ensure the data volume does not scale with the size of the problem.

A Domain Name System (DNS) [52] is a good example of how a delegation is transferred from one authority to another. DNS is a distributed system for a delegation of authority of resolving Internet domain names to different DNS servers. In a DNS, delegation starts from top-level authority called "root" (/). It delegates namespaces to various zone authorities such as com, org, net. The zone authorities can delegate different names in their zone to other child-level zones. For example, the zone com can create child zones such as google.com; yahoo.com; and so on. This hierarchy of zone authorities construct the DNS namespace. The DNS hierarchy, along with a DNS security protocol (DNSSEC) [36], guarantees authoritative and unbiased delegation of services function in one-dimensional property. However, a secure delegation of a 2D regional space is a more involved and complex process. Thus this seemingly trivial process has the following requirements to meet.

1. To secure underlying geospatial data of geographic boundaries and associated features (domains) like area, physical structures like restaurants, banks, etc.
2. To prove that an authority has a specific authority over a space to securely delegate partial or complete authority over the region, ensuring i) no overlap between delegated regions; ii) no double-delegation; and iii) no unsolicited revocation of a delegated space.
3. To avoid a trusted third party during delegation to bring greater automation for a 2D space delegation.
4. Securely and safely surrender (rollback) delegation once desired.

Details of a protocol for a domain-specific spatial delegation is discussed in Chapter 5.

1.2.3.2 E-Governance

E-governance is a service where a governing body or an organization uses information technology to deliver different services such as like issuance of citizenship, birth, migration certificates to its citizens. Other regular services that a governing body might serve to its citizens include health insurance, education allowance, and unemployment benefits, among others. In doing so, people registered for different services might be associated with a geographic location. In a demographically and geographically diverse country like the USA, efficient e-governance must embrace individual privacy of data assets; secure transaction of services, among others [34]. An attribution of a geographic location to entities such as city dwellers, disaster relief centers, voting booths, and so on could be helpful in identifying rightful and authentic entities. It would also add value in realizing greater transparency of government services. It is crucial to build secure, reliable services that facilitate an authorized and unbiased geography-based look-up services to embody spatial information with e-governance services. As an instance, a trustworthy attribution of a spatial coordinate to personal identity can serve as proof of residency in a particular political region such as State, county, city, or congressional or school district.

Similar to a Domain Name System (DNS) that binds useful information like IP address to domain names, geography-based services can be used to attach (link) a geographic location to different features such as gas stations, banks, streets, lanes, and even dynamic data like temperature, pressure, traffic, road closures, resident-related data, etc. Apart from being authoritative, responses should also be unbiased.

For instance, a query such as '*return n closest, cheapest or highest-rated gas-stations at/around a geographic location* $q(x, y)$,' must not deliberately omit any gas stations that satisfy the query.

1.2.3.3 Trustworthy Congressional Redistricting Protocol

In the context of political demarcation of a country, redistricting is a task of decomposing a political division (such as State or city) into sub-divisions (such as congressional districts, school district, etc.) to the extent that the sub-divisions meet a defined criteria such as similarity in population distribution, diversity of race or ethnic diversity, etc. Every ten years, congressional redistricting is performed to incorporate changes in population. Ideally, it should be undertaken to avoid partisan advantage or unbalanced distribution of population or race or ethnicity. Several other criteria may guide a redistricting project: compactness and contiguity of a region/division; preservation of existing political communities partisan and racial fairness [22], among others. However, as illustrated by [53], redistricting is often mired by political and legal contentions between political parties. In most cases, the problem arises from the fact that a committee of (biased) political representatives perform the task of redistricting. Redistricting performed to introduce a partisan advantage is popularly termed as gerrymandering [22, 3, 19]. Computer-assisted redistricting can alleviate such issues. However, relying on sophisticated computer system(s) to generate fair, unbiased congressional districts also raises valid questions

regarding process integrity – for the same reason that execution of classical point location algorithms can not be guaranteed.

On a different facade, given building blocks such as census blocs (or counties) for a redistricting purpose, choosing a set of blocks to be assigned to each district is an NP-hard problem [4]. Such algorithms do not lend themselves to be assured elegantly. In this dissertation, we propose a novel, trustworthy framework called Blockchain-based Redistricting Protocol (BREP) for achieving universal consensus on an (optimal) redistricting plan (among potentially thousands of districting plans). The framework is also based on the principles of the transaction-based state-transition model discussed in Chapter 2.5. Further details of the protocol will be discussed in Chapter 6.

1.2.4 Information System Security Models

Information security can be broadly classified under two security models: 1) Active Defense Model, and 2) Transaction-based state-transition Model

1.2.4.1 Active Defense Model

The active defense model approach [57, 56] is an proactive method for securing an information system. Under this model, an adversary always poses serious threats to an information system. Such threats (risks) can result in compromising key desired security assurances such as confidentiality, authenticity, and integrity of the information system. Some of the other threats can be tampering data in their storage or in

insecure channels; mal-ware or spy-ware injection in an information system; theft of credentials or critical data; or denial of service, among others.

To counter such attacks, system designers or architects attempt to design an information system that is more resistant to malware or spyware. The software engineers implement a system using robust tools. Test engineers execute test cases stringently to alleviate errors or bugs or attack points. The security engineers develop tools such as firewalls, anti-malware, and anti-virus programs. However, in times, a system might itself be behaving incorrectly. It may be due to inadvertent hardware or program bugs dormant in the platform or software modules. For example, security experts reported Meltdown and Spectre bugs in Intel microprocessors [13], and tamper vulnerable memory [27] in the recent past. Hence, a system attack model encompasses detection, identification, and removal of undesired functionality in software or hardware or intrusions in the network of an information system. This model, though, an effective as the first line of securing and defending a system, ignores issues associated with misplaced trust in hardware, personnel, complex software, etc.

1.2.4.2 Transaction-based State-transition Model

The transaction-based state-transition model [70] is a passive approach to the security of an information system. It assumes that a secure and trustworthy information system must be minimal in both software and hardware structures [54, 57]. The minimality of software and hardware provides two crucial abilities: i) an ability that

any malicious and unintended behavior would get easily detected, and ii) an ability that everyone can exhaustively verify and audit such system.

This model also principally treats an information system as a progression of a discrete set of valid system state (SS) transitions. It means that a secure system boots from universally accepted genesis SS and only verified (certified) transactions can stir consequent SS changes of the system. It implies that, under this model, an information system is a sequence of dynamic states due to permitted transactions (or invocations). A system state at time t is a snapshot of processes $\mathbb{P}$, data $\mathbb{D}$, and system specific rules $\mathbb{R}$. Securing an information system is the task of fulfilling the desired security assurances for the dynamic set of states composing a system. The major assumptions under this model are i) the correctness of $\mathbb{P}$, $\mathbb{D}$, and $\mathbb{R}$, and ii) the integrity of $\mathbb{P}$, $\mathbb{D}$, and $\mathbb{R}$ throughout the life an information system.

We identify two variants of a transaction-based state transition model for realizing secure and trustworthy geospatial services. The first variant uses a minimal hardware and software platform known as Trusted Computing Base (TCB) [44, 37, 33]. Another variant ubiquitously called blockchain network (BN), is more scalable that facilitates minimal verification of transactions in an information system. Under that model, we execute an information system on top of a tamper resistant append-only ledger (database) and distributed consensus protocol. The details of both models will be discussed in Section 2.5.

A transaction-based state-transition model, when used in conjunction with an attack model, amplifies the trust and security of an information system. More specif-

ically, in a transaction-based state-transition model, active approaches to secure components can afford to focus entirely on a minimal TCB that validates state-transitions.

1.3 Research Contributions

The research is directed towards making the following contributions:

1. Securing GIS Data Assets: The first task is to examine and implement security protocols build on transaction-based state transition models to secure GIS data assets. This task includes organizing spatial data for the USA states, counties, congressional districts, and even parcels in selected counties in specially defined structures called Ordered Merkle trees (OMTs). Some useful properties of OMTs are leveraged advantageously to implement secure, authoritative, and unbiased GIS services.

2. Assured Point Location: Design and implement an efficient data storage framework and state transition functions for the implementation of secure point location algorithms. It includes the design of blockchain state transition functions to securely manipulate secure GIS data, such as editing geographic boundary maps; and other functions for responding point location queries.

3. Authoritative Geographic Delegation: Design and develop state transition functions, and data storage framework, for geography-based delegation protocol to enable authenticated, and unbiased geographic property delegation. For example, such a protocol has utility in securely transferring ownership of land parcels, and control over geo-tagged properties such as apartments, stores, gas stations, etc.

4. A system state (SS) transition based protocol to select an optimal redistricting plan among any number of plans based on trustworthy evaluation protocols executed on a blockchain network.

1.4 Organization of the Dissertation

The rest of the dissertation is organized as follows: Chapter 2 discusses key concepts in computational geometry, scholarship on classical point location algorithms, useful concepts in cryptography, information security assurances, and some useful

cryptographic protocols. In the latter section of Chapter 2, we also outline two important system security models that are foundational to the proposed research.

Chapter 3 details a trustworthy execution of the Shamos-Hoey algorithm for detecting self-intersecting polygons, which is later used in a Secure Queryable Dynamic Map (SQDM) protocol. The SQDM protocol has been elaborated in Chapter 4. As an application of an SQDM protocol, a trustworthy geographic delegation protocol is discussed in Chapter 5. Chapter 6 outlines a secure protocol for evaluating congressional redistricting plans on a blockchain network. Conclusion is offered in Chapter 7

CHAPTER 2

BACKGROUND

> Fools ignore complexity.
> Pragmatists suffer it. Some can
> avoid it. Geniuses remove it.
>
> *A Perils on Epigrams of Programming*

Modern computing is characterized by graphical user interfaces, simulating and visualizing complex phenomena, and designing sophisticated systems. Computational geometry has greater application in other similar applications, and it has taken leaps and bounds in its development. Applications and services that rely on geographic information also exploit data and algorithms that fall into the realm of computational geometry. The field of robotics, aerial navigation, self-driving vehicular systems, and aerodynamics are the emerging fields that have much more to use of it. From abundant use cases of computational geometry, we choose the domain of geographic information and relevant services for this dissertation. We view such services from the security point of view. We propose and evaluate secure and trustworthy protocols for securing such services.

The approach for securing an information system depends on what we "wanted to secure." The models used for security solely depends on the required security assurances. For instance, active security measures focus on developing and deploying

tools to mitigate the threats on the confidentiality and integrity of a database system. Whereas, passive security approaches such as "isolating database" form a broader network that can be useful in other contexts. In this chapter, we introduce concepts of planar geometry and algorithms that are relevant to this research. It is followed by a discussion of algorithms for a planar polygon decomposition. This chapter also introduces algorithms for a classical point location problem, which is fundamental to this dissertation. In the midst of the chapter, we discuss some of the essential cryptography primitives that will be useful in later chapters, and the final section of this chapter elaborates two different information system security models that are contextual to this research.

2.1 Preliminaries on 2D Geometry

The fundamental feature in representing a geometric feature in two dimensional (2D) plane is a point feature. A point is a zero-dimensional representation of a relative location of a minuscule object. All other two dimensional objects is an assemblage of two or more point features. For instance, the two points, namely A and B, can be connected to give a one-dimensional (1D) feature called a line segment (straight) or a curve. Two dimensional (2D) polygons are closed by a set of line segments connected at their endpoints [45, 18]. By Jordan Curve Theorem [45], a polygon divides a plane into an interior, exterior, and boundary. An infinite length segment divides a plane or a polygon into two planes/polygons.

By properties, polygons can be of different types. General classes of polygons are 1) regular polygons and 2) irregular polygons. The regular polygons are bounded by equal-length segments that form a certain angle with adjoining line segments. For example, pentagon, octagon are regular polygons. All the interior angles in regular polygons are the same. The irregular polygons have no easy pattern in terms of shape or structure. They have no consistent interior or exterior angles. In both of the cases, the boundary is a sequence of line segments, not any curve. Other groups of polygons are:

- **Simple Polygons**: A simple polygon is enclosed by a sequence of segments that do not intersect other segments except the adjoining segments at their endpoints. Some operations like triangulation are possible in simple polygons. A simple polygon can be monotone or a non-monotone polygon.
- **Intersecting Polygons** : They are complex shaped polygon; the chain of the boundary segments intersect at different positions. Other common types are of polygons are convex, star-shaped polygons with holes, and so on. Figure 2.2 shows the structures of different types of polygons.
- **Monotone Polygons**: Formally, a polygon is monotone with respect to a line segment L if NO line l' perpendicular to L ($l' \perp L$) intersects the chain of the polygon at more than two points. If L is y-axis and a polygon P is monotone with respect to L, then the polygon is called y-monotone. The x-monotone polygon follows the reverse definition. A y-monotone polygon has:
  - a top vertex, see vertex A in Figure 2.1.
  - a bottom vertex, see vertex I in Figure 2.1.
  - two y-monotone chains that form the complete boundary of the polygon. In the same Figure, polygon $ABC..LA$ is a y-monotone polygon with two y-monotone chains $ABCDEFGH$ and $ALKJIH$.

2.1.1 Polygon Decomposition

Any polygon can be partitioned into two or many constituting polygons. For instance, two opposite vertices of a hexagon can be connected to construct two quadri-

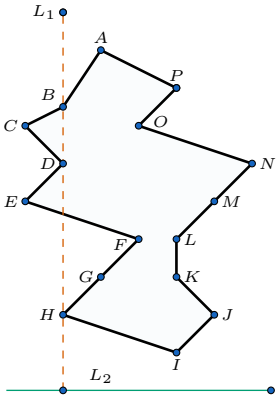

Figure 2.1

A y-monotone polygon $ABCDEFGHIJKLA$ with reference line L_1.

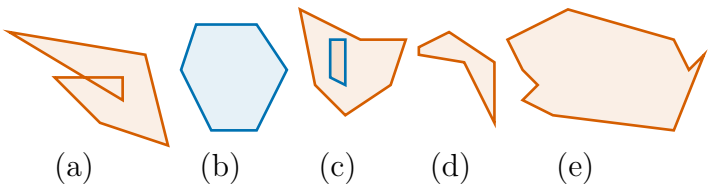

Figure 2.2

Different polygon types. (a) is a self-intersecting polygon, (b) is a regular polygon with six sides, (c) an outer polygon contains an inner polygon called hole, (d,e) simple, irregular polygon)

laterals inside a hexagon. Similarly, two opposite pairs of vertices can be connected to construct four non-overlapping triangles inside the polygon. This task of disintegrating a bigger polygon into smaller polygons, however, does not always produce an applicable decomposition. We discuss some decompositions that are useful to solve the point location problem in the following section.

- **Monotone Polygon Decomposition:** The process of dividing a simple polygon into a set of non-overlapping monotone polygonal pieces is called monotone polygon decomposition. Decomposing a simple polygon into a set of monotone polygons. A monotone polygon can be optimally decomposed into more finer polygons like a triangle. Algorithms such as sweep-line efficiently decompose a simple polygon in $O(nLogn)$ time and using up $O(N)$ space.
- **Polygon Triangulation:** Filling a polygon by maximal non-overlapping pieces of triangles is called a polygon triangulation. A set of non-intersecting diagonals are inserted between suitable polygon vertices to insert triangles inside a polygon. Such diagonals fall entirely inside the interior of the polygon. As different sets of diagonals can be inserted to fill up the same polygon, triangulation is not unique to a polygon. However, any triangulation is aimed to reduce complex shapes to a collection of simpler triangles [24]. Some of the areas for application of triangulation include 1) Visibility and resource optimization, graph coloring, 2) robotics and path planning, 3) generation, 4) point location, among others. Every simple polygon can be decomposed into non-overlapping triangles. Algorithms to triangulate a monotone polygon can run in $O(nlogn)$[24].

2.2 Classical Algorithms for Point Location Problem

The algorithms for solving classical point location problem use two fundamental techniques: 1) Polygon Decomposition, and 2) Search. Having discussed fundamental concepts on 2D polygons and 2D polygon decomposition techniques, let us discuss on different classical point location algorithm for a polygon with n vertices.

1. Slab-Decomposition Method: It is a simple algorithm for the point location problem. With this technique, we divide a polygon into vertical slabs whose vertical sides passes through each of the vertexes of the polygon. To locate a point, first, a vertical slab that contains the point is identified. Second, all the

segments that fall inside the segments are sorted in increasing y-ordinates of the end-point. Finally, the region between the two segments, which encloses the y-value of the query point, is identified as a solution. This algorithm has query complexity of $O(logN)$, and space complexity of $O(n^2)$. Sarnak and Tarjan improved this technique to achieve $O(n)$ space using a dynamic search tree called *persistent search tree* [5].

2. Triangular Refinement Method: Point location by triangular refinement technique was invented by D.G Kirkpatrick in 1983 (as cited in [5]). In this method, a simple polygon is repeatedly decomposed into non-overlapping triangles. This means that finding a triangle that contains a query $q(x, y)$ solves the point location problem. A rooted search tree known as a directed acyclic graph (DAG) is maintained, which helps to narrow the search for a triangle that contains the query point. With this method, a query time of $O(logN)$, space complexity of $O(n)$ is obtained.

3. Incremental Trapezoid Decomposition Method: It was developed by Mulmuley [5]. In this method, a polygon is decomposed into non-overlapping $O(N)$ trapezoids with two parallel edges. However, a search tree DAG with expected depth $O(logN)$ is a trapezoidal map that is incrementally built up as a polygon is decomposed into trapezoids. A top-down traversal in the DAG is made to locate a leaf node (a trapezoid) containing the query point to solve a point location problem.

4. Segment Tree Method: Dynamic point location allows the insertion and deletion of vertices or edges to change the subdivisions. Algorithms for such problems must optimize insert time, delete time, storage requirement, and query time. Overmars [17] reported a method that uses $O(nlogn)$ space and $O(log^2 n)$ query and update time. The method used a segment tree for storing the segments of a map, and a collection of balanced trees, called boundary trees, representing the boundaries of the regions of the map.

5. Chain Method: Fries and Mehlhorn [30], used an approach based on the static chain method to create a data structure that has $O(n)$ space, $O(log^2 n)$ query time and $O(log^4 n)$ amortized update (insert and delete edges or isolated vertices) time. Insertion only time is $O(log^2 n)$. Tamassia and Preparata provide two methods for a dynamic point location in monotone and convex subdivisions [29, 51]. The first method [29] is for a monotone subdivision based on the chain method [39]. The data structure requires $O(n)$ space. The update operations (insertion of monotone chains of edges and vertices on edges) can be performed in $O(logn)$, and $O(log^2 n + k)$, k is a number of inserted/deleted edges. The query time is $O(log^n)$. The second method is for convex subdivision based on the trapezoid method. Their method achieved space $O(N+nlogN)$, query time $O(logn + logN)$, and update time $O(lognlogN)$, where N size of fixed set of horizontal lines where each vertices lie.

2.3 Preliminaries on Cryptographic Algorithms

This section introduces the key concept of modern cryptography. During the discussion, we follow useful notations and shorthands as: M is an *n-bit* message such that $M \in \{0,1\}^0$, an encrypted version of a message M is denoted by $C \in \{0,1\}^*$; acronym PRF for a pseudo-random function; symbols '$\cdot$' or '$\|$' between two operands indicates concatenation operation of the operands; assume that *Alice* and *Bob* are entities that communicate through an insecure channel; *Eve* is an adversary who attempts to tamper or sniff the messages (data) that transfer between *Alice* and *Bob*. The pronouns *she* refers to *Alice*, and *he* is used to refer to both *Bob* and *Eve*.

2.3.1 Security Assurances

Unlike the classical practice of cryptography, modern cryptography work is not just about hiding information. It provides ways to construct advanced tools, and strategies to achieve well -defined security goals or assurances [55]. In modern times, computer systems consume large data storage, and data processing works. People use a digital communication network to exchange information through a computer network like the Internet. Both computer systems and a digital communication network must provide information security requirements such as data privacy, integrity, origin authorization, among others. Depending upon applications and services, other security goals such as response completeness, and non-repudiation are also key security requirements. An information system and a communication network must abide by certain rules in processing data and in exchanging information between parties

(e.g., users, network-node, external process, etc.) to deliver specific security requirements. The set of rules that processing nodes (such as computer, services, routers, switches) and communicating parties follow to meet the defined set of information security requirements constitute a security protocol. The building blocks of a security protocol consists of a set of cryptographic tools, underlying non-cryptographic assumptions, and cryptographic protocols [55]. The underlying tools that confirm certain relations about input and output can be termed constitute cryptographic protocols. Essential cryptographic protocols are provided by cryptographic primitives. These non-cryptographic assumptions provides context to the input and output of the cryptographic protocols. Before presenting details on cryptographic primitives, let us elaborate on some of the critical security requirements of digital systems and the communication as defined in [35, 42].

- Confidentiality: Let us assume that Bob and Alice exchange messages through an insecure channel. The property of their message confidentiality means that the content of messages that relay between Alice and Bob is not seen or known by any third party (example, adversaries to Alice or Bob or both; public audience, etc.). This property is synonymous with privacy or secrecy of communication and storage of data. In other words, confidentiality affirms: "The content of data in storage or in the communication channel is known only by the owners of the data."

- Integrity: The property of data integrity also extends to the data in a storage and the data in a communication channel. This property ensures that the original data is not altered or tampered by unauthorized parties. To assure data integrity, an authorized owner must have an ability to detect any unauthorized data modification (such as insertion, deletion , or substitution can alter data)[42, 35].

- Authentication: Let us consider that Alice and Bob are communicating through an insecure channel. Alice must ensure that she is indeed in communication with Bob she knows, and no other entity is emulating Bob. The same applies to Bob as well. Thus Alice must authenticate her destination entity before and

during her communication with Bob. The property of origin authentication allows sender and receiver to identify each other before initiating communication.

- Completeness: Assume that Alice makes a query q to Bob. Alice expects to receive a complete set of valid responses, r. However, Bob might malfunction to send only a partial response to Alice's query. The property of completeness allows Alice to verify if she receives a complete set of valid responses from Bob. For instance, a database range query must result in complete tuples inside the query range.

- Non-repudiation: The property of non-repudiation ensures the identity of the origin of information without leaving a chance to deny the act in the future. For instance, if Alice repudiates sending a message to Bob, an independent third party, possibly a judge in a court, must generate proof that affirms if Alice indeed sent the message to Bob.

2.3.2 Digital Signature Schemes

A digital signature scheme of G, $\textsc{Sign}_{(\cdot)}(\cdot)$, $\textsc{Verify}_{(\cdot)}(\cdot)$ is a suite of three algorithms that ensures three critical properties of message integrity, authentication, and non-repudiation [32]. Let us suppose that *Alice* wants to disseminate a message to Bob. Bob must be able to verify that the message comes from *Alice*. The digital signature provides a credible system for the purpose.

1. G is a probabilistic polynomial-time key generation algorithm. Similar to G in public-key cryptography, it establishes a private key K_r^A and a public key K_u^A associated with the private key. *Alice* uses G to obtain a private key K_r^A, and a public key K_u^A. *Alice* publishes her public to sign a message M she sends to Bob.

2. $\textsc{Sign}_{(\cdot)}$ is a signing algorithm. *Alice* computes a digital signature $\sigma_A \leftarrow \textsc{Sign}(M,K_r^A)$. She then transmits ($M$, σ_A) to Bob (or other potential recipients.)

3. $\textsc{Verify}_{(\cdot)}(\cdot)$ a verification algorithm. Upon receiving (M,σ_A) from *Alice*, Bob uses algorithm $\{0,1\}^1 \leftarrow \textsc{Verify}(M,\sigma_A)$ to verify the authenticity of M by checking that σ_A is a legitimate signature on M with respect to the public key K_u^A.

Popular schemes for digital signature are RSA Digital Signature, ElGamel Digital Signature, Elliptic Curve Digital Signature Algorithm (ECDSA), among other. However, GOST R 34.10-2012, ECDSA is a popularly used digital signature algorithm with several merits over other algorithms.

Security Aspect of a digital signature: A digital signature is analog to a physical handwritten signature. It is a binding that represents a document to its signer. Two properties of a digital signature ascend its high value. Firstly, nobody except a document owner can sign the document. Secondly, anyone can verify the validity of a signature, and associate a signature to its signer. It is similar to the MAC protocol. However, it has more merits than MAC protocol. Unlike MAC, a digital signature can be shared to multiple recipients without sharing a secret key. The digital signature scheme also provides a very important property of non-repudiation. It means that once *Alice* signs and publicizes her public key, and then signs a message, she cannot later deny that she sent. Digital signature algorithms are not expensive, and it is infeasible to determine a private key K_r^A, given the corresponding public key K_u^A. In this research, we will denote *Alice*'s signing a message M with her private key K_r^A by $\textsc{Sign}_{K_r^A}(M)$, and Bob's verifying a signature with corresponding public key K_u^B by $\textsc{Verify}_{K_r^B}(M)$.

2.3.3 Cryptographic Hash Functions

A hash function is a one-way function, H, that maps an input string M of any size to a fixed size value h, commonly called hash value or a digest for M. H is a

deterministic function that can efficiently compute a digest for an input string of any size [55, 31]. Ideally, a cryptographic hash function must satisfy the following three properties:

1. Collision resistance: Two messages m_1 and m_2 that map to the same hash-value $h = H(m_1) = H(m_2)$ identify a collision for a hash function. For a collision-resistant hash function, it is hard to find a collision. Given d bit digest $h = H(m_1)$ of a message m_1 produced by hash function H; it takes 2^d attempts to find a pre-image $m_2 \neq m_1$ such that $h = H(m_2)$. Thus it is impractical to find collision resistance for a hash function H that produces a higher bit digest.
2. Pre-image resistance: Given a hash value h for a message m, it should be hard enough to find other message $m' \neq m$ such that $h = H(m')$. The only way to determine two colliding pre-images must be brute-force, where two random messages m and m' are chosen to verify $(H(m) == H(m'))$. For a function H that outputs d bit digest (hash value), the minimum of $2^{d/2}$ brute-force attempts are required to find the colliding pre-images. This property makes a cryptographic hash function purely a one-way function. This property is also known as the data hiding property of a hash function.
3. Second Pre-image resistance: Given an input message m_1, it is computationally infeasible to find a different input message m_2 such that $H(m_1) = H(m_2)$. Given n-bit hash values, the time-bound for second pre-image resistance is 2^n. The functions that do not confirm this property are prone to the second pre-image attack.

Cryptographic hash function has utility in many applications such as digital signature and verification, message authentication codes (MACs), database searching/indexing, data corruption detection, (integrity), error checking, duplicate file identification, document time stamping, among others. Popular hash functions includes MD5SUM, SHA-1, RIPEMD-160,WHIRLPOOL, SHA-2, SHA-3, BLAKE2, among others.

Security Aspect of hash functions: Let us consider that *Alice* wishes to check the integrity of a large file she uploaded to cloud storage (e.g., Google Drive, Amazon,

Microsoft, etc.) A simple way she can do is to download the file locally and compare it with the original file in her machine. This method obviously conflicts the purpose of uploading files in the cloud. The collision-resistance property of a hash function can solve *Alice*'s problem elegantly. The property allows one to safely assume that a hash function produces a fixed-size digest h for an input message M. *Alice* can compute and remember the hash value of her original document before she uploads her document to the cloud. She can download the uploaded document and again compute its digest. The two hash values can be compared if they match. This infers that a message is uniquely related to its hash value. Moreover, knowledge of digest h does not reveal an input message M. This property is useful in constructing a commitment value for any message. For instance, assume that *Alice* sends a message M and its digest d to Bob. On receiving a message M' and digest value d; Bob computes his version of digest of the message and compares with Alice's digest to be confident that Alice committed to message M.

2.3.4 Hash accumulator

Given a hash function $h(.)$ as defined in previous section, it can be applied in a sequence to a list of growing values $v_0, v_1, v_2...v_n$. A hash accumulator α_i for a hash function $h(.)$ applied sequence of i values in a given list is computed as:

$$\alpha_0 = v_0, \alpha_1 = h(\alpha_0||v_1), \alpha_2 = h(\alpha_1||v_2), ...\alpha_n = h(\alpha_{n-1}||v_{n-1}) \tag{2.1}$$

The accumulated hash α_i is a cryptographic commitment to all values in a sequence $v_0, v_1 ... v_i$. As discussed earlier, the property of a hash function $h()$ ensures that it is impractical to determine any sequence of values different from $v_0, v_1, .., v_i$ that satisfies accumulated hash of α_i. Bitcoin and Etherium ledger are suitable examples of an accumulated hash. A sequence of blocks (containing multiple transactions) are connected in a chain by using an accumulated hash, which is a commitment to all previous blocks of transactions in the Bitcoin/Etherium ledger.

2.4 Data Structures

In this section, we discuss some of the data structures that are useful in most of the secure protocols discussed as part of our researches.

2.4.1 Dictionary

A dictionary is an assemblage of key-value pairs, where keys are unique in face value in a collection [11]. For example,

$$(2, 5), (4, 7), (456, 21), (5, 0), (32, 6)$$

is a collection of items in a dictionary where keys are: 2, 4, 456, 5, and 32; and values are: 5, 7, 21, 0, 6. A dictionary entry $(k, 0)$ with value 0 can be specially treated as a placeholder entry. A dictionary such $\{(k_0, 0), (k_1, 0), .., (k_n, 0)\}$, where all keys' values set to zero, can be interpreted as a place holder dictionary. The value for a specific dictionary key k can be modified at a later time, obeying some domain-

specific update rules. A sorted dictionary is a set $\{(k_0, v_0), (k_1, v_1), .., (k_n, v_n)\}$, where keys $k_0, k_1..k_n$ are arranged according to temporal or numerical orders. It supports the insertion or deletion of new entry (k', v') so that the dictionary always remains sorted. In general, a dictionary D is a mapping $\{(k_0, v_0), (k_1, v_1), ..., (k_n, v_n)\}$, where key k_i is assigned to a value v_i.

2.4.2 1D-Look-up-table (1D-LUT)

A one-dimensional Look-up-table (1D-LUT) is a collection of entries of the form $[k, k_n, v]$[11]. The first element k is a key or index for which value is v, and k_n is key/index of the next following entry. However, all entries in an LUT collection

$$\{[k_0, k_1, v_0], [k_1, k_2], ..[k_n, k_0, v_n]\}$$

can be interpreted as a definition of a piecewise function $f(k)$ over all $k \in \{k_0, k_1..k_n\}$. For example,

$$[5, 8, 256] \quad [8, 13, 46] \quad [13, 37, 0] \quad [37, 45, 1234] \quad [45, 5, 0]$$

is a 1D-LUT collection with key/index $5, 8, 13..45$. The value of a function $f(k)$: $(5 \leq k < 8)$ is $v = 256$; the value of function $f(k) : 8 \leq k < 13)$ is $v = 46$; and so on. A collection is complete if a piece-wise function is also defined for an interval $k_n[, k_0[$, where k_n is maximum domain, and k_0 is minimum domain value in the collection. A value of such entry can be permanently set to 0, which implies that the functional

value of $f(k) : k_n < k; k_n < k_0$ is undefined over that range.As the value of all keys greater than 45; and all keys smaller than 5 is set to be 0, the above 1D-LUT is a complete LUT.

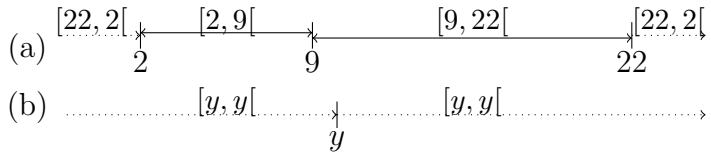

Figure 2.3

Intervals of a 1-D OMT collection with (a) 3 intervals/leaves and (b) 1 interval/leaf

Intuitively, a 1D LUT is a collection of intervals along a number line. Consider a collection of items $\{[2, 9, u_1], [9, 22, u_2], [22, 2, \phi]\}$. Together, the items represent three intervals [2, 9[, [9, 22[and [22,2[in a number line as shown in Figure 2.3 (a). The values u_1, u_2 and ϕ are the values associated with the intervals. Specifically, these intervals define three different piece-wise functions $u_1 = f_1(x) : 2 \leq x < 9$, $u_1 = f_2(x) : 2 \leq x < 9$; and $u_1 = f_3(x) : 2 \leq x < 9$. We can notice the last interval [22,2[which is a 'wrapped around' interval that represents all the undefined values exceeding 22 and less than 2. A complete collection must have one such interval. An existing interval can be split into two to insert a new interval; and two adjoining

intervals can be merged if they satisfy certain conditions. In general, a complete LUT L is an n-entry collection of the form:

$$\{[k_0, k_1, v_0], [k_1, k_2, v_1], ..., [k_j, k_{j+1}, v_j]...[k_n, k_0, 0]\} \tag{2.2}$$

However, a 1D-LUT can be interpreted as a simple dictionary collection with a linked key for each key. In a dictionary definition of an entry (k, k_n, v), the value v is associated with unique point k on a number line, which will be followed by definition of next point k_n on the line. For instance, a dictionary interpretation of the entry $[5, 8, 256]$ in the above collection would be: A fixed point $k = 56$ on a number line is associated with a value of 256, while the next point that follows it is $k_n = 8$.

2.4.3 2D-Look-up-table (2D-LUT)

Similar to a 1D definition, an LUT can also be defined over a 2D space. It defines a functional value over a uniform rectangular 2D-space bounded by x_l on the left; x_h on the right (exclusive); y_l on the bottom and y_h on the top (exclusive). Syntactically, it can be represented by a 5-tuple entry:

$$[x_l, y_l, x_h, y_h, v]$$

It can also be written as $(x_l, x_r) \times (y_l, y_h), v[$, which is functional definition over a rectangle $(x_l, x_h) \times (y_l, y_h)$. For example consider a collection, B, with three 2D-LUT entries,

$$B = \{[1, 1, 2, 2, R_1], [1, 2, 2, 3, R_2], [1, 3, 2, 1, R_3], [2, 1, 1, 1, R_0]\}$$

It defines two piece-wise functions over two unit area rectangular regions $(1, 2) \times (1, 2)$, and $(1, 2) \times (2, 3)$. The last entry $[2, 1, 1, 1, R_0]$ is a "wrap around" function definition that completes definition over vertical column that extends before 1 and after 2 along x-dimension.

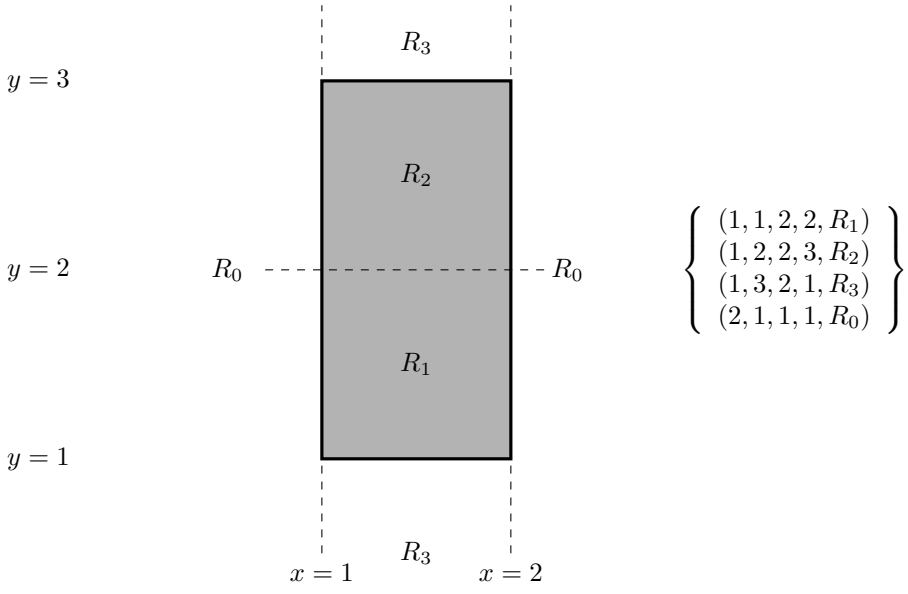

Figure 2.4

Rectangular regions represented by three 2D-LUT entries in B.

Intuitively, a value v for a 2D-LUT entry $[x_l, y_l, y_h, x_h, v]$ represents value for every point that falls in the region bounded by a rectangle $(x_l, x_h) \times (y_l, y_h)$. An entry such as $[x, y, x, y, v']$ represents an entire 2D plane assigned a value v'.

2.4.4 Merkle Hash Tree

A Merkle (1987) hash tree [43] is a binary tree of hash values constructed on top of data records in a collection. Consider a collection of records, $C = \{L_0, L_1, .., L_N\}$. A Merkle tree $\mathcal{T}$ for the collection C is constructed as firstly, apply cryptographic hash functions to all N records in C to produce N hashes $\{v_0^3, v_1^3 ... v_N^3\}$ at level-3; each of these hashes becomes leaf values in the tree $\mathcal{T}$. These N hashes are paired to produce $N/2$ hash pairs (v_i, v_{i+1}). Again, a cryptographic hash function is applied to each of the pairs to produced $N/4$ parent hashes $\{v_0^2 = h(v_0^3, v_1^3), v_1^2 = (v_2^3, v_3^3), ...\}$ at level-2. This process is repeated until a single root value σ is produced at the top level-0. The root value is called a cryptographic commitment to all the records in C. Figure 2.5 describes the construction of an OMT for database records

$$\{L_0, L_1, L_2, L_3 .. L_7\}$$

Mainly, a Merkle tree has utility in producing a single, succinct address to virtually any number of data records. Any change in a data record can be detected by reconstructing a root hash value to compare with the previously produced one. A Merkle hash tree can produce two types of proofs about a record r in a collection:

1. proof of membership: Given a record L_1, a minimal proof of membership that $L_1 \in C$ can be efficiently produced using a Merkle tree. For example, as shown in Figure 2.5 , to prove that L_1 exists in C, it is sufficient to supply a verification objects, $\mathcal{VO}$s, containing hash values $[v_0^3, v_1^2, v_1^1]$. The receiver computes sequence of hash operations as: $v_a = h(v_0^3, v_1^3)$, and hence $v_b = h(v_a, v_1^2)$, and $\rho' = h(v_b, v_1^1)$. If $L_3 \in C$, then $\rho = \rho'$ must hold true. At the same time, one can trust the integrity of the leaf L_1 and its complementary values. Let f_m be a function that computes a sequence of hash operations on a verification object $\mathcal{VO}_\rangle$ for a given input leaf node L_i (data record), to produce a root value σ'. It is given by:

$$\rho' = f_m(h(L_i), \mathcal{VO}_i) \tag{2.3}$$

From 2.3, the fact that $f_m(h(L_i), \mathcal{VO}_i)$ produces root σ of a tree shows that leaf L_i and $\mathcal{VO}$s must exist before the root σ was computed. Thus, $\mathcal{VO}_{L_i}$s is a proof of membership of leaf record L_i.

2. proof of integrity: If proof for (1) is established, then an integrity of record L can be safely ascertained.

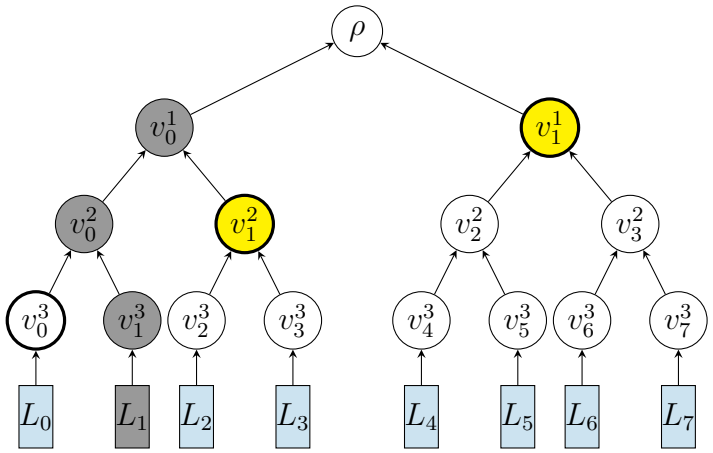

Figure 2.5

A binary hash tree with 8 leaves. For the data element L_1, nodes on the path up to the top ancestor are $p = \{L_1, v_1^3, v_0^2, v_0^1\}$ (dark filled). The values stored in the nodes are the hash function of the values in the child nodes beneath one level. The sibling nodes of the nodes on the path p form a set of complementary nodes $\mathcal{VO}_{L_i} = \{v_0^3, v_1^2, v_1^1\}$ (thick circled nodes), also called complementary hashes to record L_1.

Given N records in a collection, the height (number of levels) in a Merkle tree is $logN$. The total number of hashes in the tree is $(2N-1)$ or $(2^{logN+1}-1)$; the size of verification objects $\mathcal{VO}$s is $logN$; at most $O(logN)$ hash operations are performed to produce root of a Merkle tree given $logN$ size verification object. For instance, a Merkle tree with a billion records produces only 30 hash-values as $\mathcal{VO}$s; 30 hash-operations are required to reproduce a root of the tree to verify the existence of a leaf record. A Merkle tree, however, does not support proof of non-membership in a collection. However, different modifications of a Merkle tree such as an Ordered Merkle tree permits to produce proof of non-membership of a record to a set.

2.4.5 Ordered Merkle Tree (OMT)

Similar to a general Merkle tree ,as cited by [44, 11], an OMT is a binary Merkle hash tree whose leaf nodes are a collection of data items that are maintained in a specific order. This means that the position of a data element in the tree is known. The data elements can be typically ordered according to numerical ordering, lexicographic ordering, and chronological ordering. Let $C = \{L_0, L_1, ...L_N\}$ be a collection of data entries stored in leaf nodes of an OMT. Then, each leaf L_i in an OMT can take one of the 3 types of interpretations:

> **Interpretation Type 0**: a simple dictionary entry (k, v) commonly denoted as (k, v) with usual semantics discussed in section 2.4.1. It can also be commonly denoted as (k, k_n, v) or $(k, v)_{k_n}$ with usual semantics discussed in section 2.4.2
>
> **Interpretation Type 2**: an interval interpretation of 1D-LUT $[k, k_n, v]$, commonly denoted as $[k, k_n, v]$ or $[k, v]_{k_n}$ with usual semantics in 1D discussed in section 2.4.2

Interpretation Type 3: an interval interpretation of 2D-LUT $[x_l, y_l, x_h, y_h, v]$, commonly denoted as $[x_l, y_l, x_h, y_h, v]$ or $[x_l, y_l, v]_{x_h, y_h}$ with usual semantics in 2D discussed in section 2.4.3

Altogether, every record key is unique in a collection, and all leaf nodes stores data of a single leaf type. Since a collection in an OMT must be complete, there exists an $L(k', k'_n, 0)$ such that condition $k' > k'_n$ identifies the lowest key k'_n, and the highest key k'. A singleton set C (cardinality $|C| = 1$) contains a lone leaf of any four types: a Type I leaf $(k, k, 0)$; a Type II leaf $[k, k, 0]$; and Type III leaf $[x_l, x_l, y_l, y_l, 0]$–with usual interpretation.

Several permissible operations such as insertion of new leaf, updating the value of an existing leaf, or deleting a leaf, checking enclosure, provided a dynamic state of an OMT. The operations are performed using specific rules and verification protocol. For instance, a leaf with key j can be inserted to an OMT only if no leaf such as $[j, k_n, v_j]$ currently exists in the OMT. It is performed by verifying proof of the non-existence of such leaf. Actually, the proof for non-existence is a simple proof of the existence of a leaf (i, i_n, v_i) in the OMT such that:

- $i \leq j \leq i_n$; if $i_n > i$ OR
- $j < i_n \leq i$ or $i_n \leq i \leq j$; if $i_n \leq i$

For example, proof of the existence of $(102, 120, v_0)$ is a proof that a key-value pair $(102, v_0)$, which also sufficiently proves that keys $[103, 104...119]$ does not exist in an OMT. Similarly, an existence of leaf such as $(120, 102, v_1)$ is proof that minimum key in the collection is 120, and the maximum key is 102. A proof that key-value $(102, 102, v_0)$ is proof of singleton item (with key 102) in the collection.

2.4.5.1 Prover-Verifier Protocol

In a two-party (prover-verifier) Merkle hash tree, an untrusted prover maintains a collection C of all $N = 2^d$ records; and a Merkle hash tree hash nodes for the collection. To prove an existence of a leaf $L_k : [k, k_n, v]$ in an collection $C = \{L_0, L_1, ..L_k, ...L_N\}$; the prover supplies $logN$ verification objects $\mathcal{VO}$s produced for the leaf L_k; the leaf L_i to a potential verifier; and signed root ρ of the OMT for C. The verifier computes a potential root value σ' given by 2.3. An equivalence of σ and σ' would guarantee:

- existence of a leaf $L_i : [k, k_n, v]$ in an OMT $\mathcal{T}'$ with dynamic root σ–which we denote as $[k, k_n, v] \in \mathcal{T}'/\sigma$ or $[k, v]_{k_n} \in \mathcal{T}'/\sigma$ or simply $\mathcal{T}'/\sigma : [k, v]_{k_n}$;
- integrity of leaf $[k, k_n, v]$;
- integrity of $logN$ verification objects $\mathcal{VO}$s in an OMT $\mathcal{T}'$ with current root σ–which we denote as $\mathcal{VO}_{L_i} \in \mathcal{T}'/\sigma$.

Insertion and Deletion of a Leaf in an OMT

An insertion of a leaf node in an OMT is an iterative process. Initially an OMT is considered to be an empty collection $C = \{\}$, with root of the OMT $\mathcal{T}'$ as $\sigma = 0$. The first insertion occurs by inserting a leaf that defines first placeholder **type II** leaf $[k, k, 0]$, where k is an arbitrary key with an initial value of 0. All subsequent new key-value pair (j, v_j) (or new leaf $[j, , v_j]$) can be inserted by taking the following steps:

- verify that leaf $[j, v_j] \notin \mathcal{T}'/\sigma$;
- find an existing leaf $L_s : [i, i_n, v_i]$ such that
  case-1: if $i < i_n$, $i < j < i_n$ OR
  case-2: if $i_n < i$; $i_n \leq i < j$ or $j < i_n \leq i$

Intuitively, the new key j must be either enclosed by a split leaf, or fall/lie on either exterminates of the existing OMT space.

- split L_s into two leafs $L_l[i, j, v_i]$ and $L_r[j, i_n, v_i]$ (a placeholder); in case of **Type I** (dictionary interpretation) leaf, L_r is $[j, i_n, 0]$
- update placeholder leaf L_r as $[j, i_n, v_i \rightarrow v_j]$
- update $h(L)$ as: $h(L) \leftarrow h(h(L_l), h(L_r)$; and update all ancestors of $h(L)$ in the OMT. The current root σ will be modified to σ'.

Example-1: Consider an OMT $\mathcal{T}$ with root value of σ_T and leaves of 1D-LUT collection:

$$C' = \{[21, 27, a], [27, 33, b], [33, 47, c], [47, 55, d], [55, 60, e], [60, 27, 0]\} \tag{2.4}$$

The existence of leaf $[60, 70, 0]$ shows that collection C' is a complete 1D-LUT collection, with the lowest key 27, and highest key 60. To insert a new key-value pair $(36, f)$ into the collection, the first we find and verify that an existing leaf that encloses new key 36 is $L_s : [33, 47, d]$. Secondly, we split L_s into two leaves: $L_l[33, 36, d]$, and $L_r[36, 47, 0]$ (placeholder leaf). Thirdly, update leaf L_r's value to 4 so that the leaf becomes $[36, 47, 0 \rightarrow f]$. Lastly, the hash value for original leaf L_s is updated by hashes of two new child leaves, L_l and L_r: $h(L_s) \leftarrow h(h_{L_s}, h_{L_r})$. Finally, all the ancestors of $h(L_s)$ are updated by new hash values to update the old root σ_T of OMT $\mathcal{T}$ to σ'_T.

A similar suit follows for inserting Type I, and Type III leaf in an OMT for collections of such types of leaf structures. By now, it is helpful to highlight the significances of an OMT:

1. To produce a single content address for an unlimited number or size of records. It is called Merkle Tree root hash, if signed by an owner, an integrity of underlying data records can be assured. Hence, OMT's root facilitates to track the records in a dynamic database of any size maintained by untrusted parties.

2. It can be used to produce proofs of integrity, existence, or non-existence of a record in a data set. Any verifier can use the proofs to verify an assertion from an untrusted server.

3. Incremental update to a leaf node in a Merkle tree can facilitate: i) updating an existing leaf $L \leftarrow L'$; ii) inserting a new leaf L' to update $h(L) \leftarrow h(h(h(L), h(L'))$; and iii) delete an existing leaf L' to $h(h(h(L), h(L')) \leftarrow h(L)$. All these operations must follow updating $LogN$ hashes up the Merkle tree to update an old root $\sigma \leftarrow \sigma'$. Hence it is evident that a Merkle tree can be used to track the integrity of unlimited data records C.

   Ordered Merkle tree or other derivatives of Merkle tree are the backbones of distributed systems like Bitcoin, and BitTorrent, among others [12].

2.5 Information System Security Models

The security of an information system must be based on some fundamental assumptions regarding a computing platform's nature and operations. For instance, it is assumed that in a general-purpose computing machine, its a temporary memory, a central processing unit (CPU), and input/output channels are resistant to any malicious attacks [44]. It is also assumed that any driver or system programs are correctly implemented to produce an accurate output for an application program. However, a litany of contemporary system failures and security breaches falter the assumptions. It suggests that system security approaches must stretch beyond the week assumptions. Of all passive security measures, a *system integrity model* based on Trusted Computing Base (TCB) has been popularly investigated and used to cater to the security requirements such as access control, privacy, and system integrity. A

TCB has been suggested to be essentially useful in securing systems with complex parallelism, multi-threading, and connectivity [58].

2.5.1 Hardware-Based Trusted Computing Base (TCB)

Butler et al. [37] defined TCB as '*a small amount of software and hardware that security depends on.*' Trusted Computer System Evaluation Criteria (TCSEC), also known as the Orange Book, recognizes TCB to have an ability "to enforce correctly a unified security policy" for an information system [25]. The principal aim to have a TCB with minimal hardware and software is to reduce the surface of malicious attacks, and to minimize the probability of bugs or defective features [33]. A hardware TCB is also viewed as a transparent computing module. It is reasonable because key security assurances such as integrity and correctness can be verified and agreed upon publicly. This implies that the computing platform's strength and vulnerabilities can be publicly assessed; its codebase can be exhaustively examined and audited. It is, thus, anyone can place a higher level of trust in its operations and outputs. A TCB has the following key features:

1. It operates on low complexity hardware.
2. It has a minimal write-proof and read-proof memory.
3. It has limited state registers that leverage for storing critical data elements such as encryption and decryption keys, and private key(s).
4. The external sub-systems can invoke verifiable minimal operations (methods) stored in a TCB for the sub-system. A TCB executes the operations only after verifying pre-conditions defined for the invocation. The execution can result in state changes, which can be certified by issuing certificates to the external sub-systems. The TCB can optionally store the state change in its internal registers to certify future method invocation.

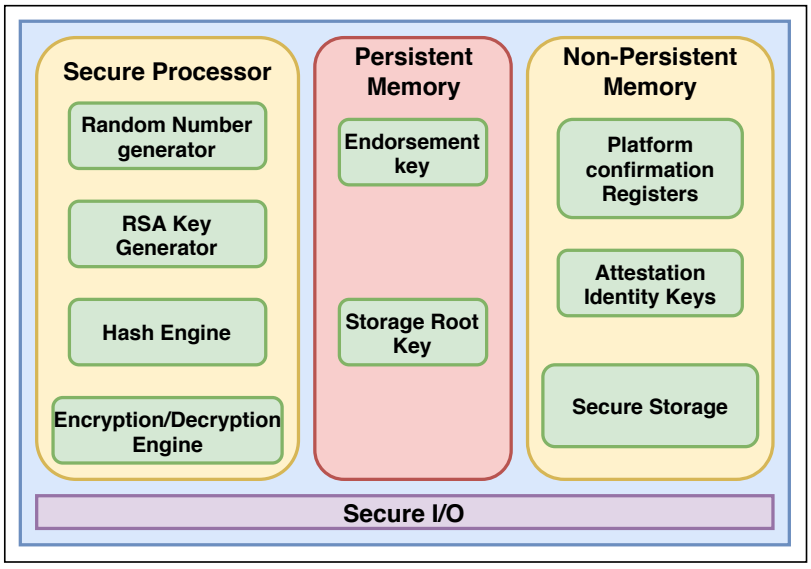

Figure 2.6

The general architecture of a TCB consists of i) Secure Processor, ii) Persistent Memory of limited size, and iii) Non-persistent Memory with limited size. The secure Input/Output channels facilitate communicating critical data with the process outside the TCB.

The general architecture of a TCB is shown in Figure 2.6 As [67] posits, usage of a TCB to amplify the security and reliability of a system comes with challenges of minimizing its storage and computational requirements. It is, therefore, minimizing a TCB for an information system that has been an emerging field of research in computer science. Agencies like the National Institute of Science and Technology (NIST) and the Department of Defense (DoD) recommended building a system confirming the TCB requirements [25]. Microsoft Inc. has long been using its TCB architecture to secure an operating system (OS) and the OS's boot loader. Specifically, it uses the module for access control and authentication. Doctorate level researches focus on building secure systems based on TCBs [41].

2.5.2 Consensus-based Blockchain Network

A set of independent computing nodes in a distributed system can reach an agreement on uniqueness and correctness of canonical initial state, S_0, of an information system I at time t_0. An atomic state-transition function f_i is invoked in a (an optionally peer-to-peer) broadcast network to change the current state of the system to a new state S_1. The network nodes independently verify the validity of the input transaction; and execute the function to produce an output. If not all, the majority of them can again agree upon the correctness of the function's final result causing state change from S_0 to S_1. The new state S_1 is added into an immutable, append-only public ledger(record) B for consequent future state changes $S_2..S_n$. In this model, a single atomic function causes state change of an information system.

Thus a system can be considered as a sequence of valid, discrete state transitions. The mechanism by which a distributed nodes concur upon state changes of a system is a consensus protocol. The network protocol that uses an immutable ledger (also called blockchain) is called a blockchain network (BN). Currently, blockchain network protocols such as Bitcoin and Etherium are instances of consensus-based blockchain networks [12, 58, 46]. They use Proof of Work (PoW) and Proof of Stake (PoS) to reach a consensus on the integrity of process output. Let us explain more about this infrastructure by an example. Consider Bob wishes to transfer some units of currency to Alice living in other countries. Under the conventional system, Bob can use remittance services such as Western Union or Paypal.com to transfer an amount x to Alice. At this point, it is essential to understand why Bob has to use a remittance service for transferring value across the country. The answer to this question lies in a principal notion of trust. Both Bob and Alice put trust in a mediator like Western Union or Payal. Both of their trust is due to the fact that the mediator grantee payment of money to Alice. It also holds Bob's and Alice's accounts information centrally to perform transaction verification such as: if $x <= (\sum_{Bob} + c)$, where $\sum_{Bob}$ is Bob's current account balance, and c is a small service charge. Several problems inherent to this mode of value transfer process include

1. Existence of trusted third party authorities or intermediaries,
2. data centralization,
3. Higher service fees (given the minimal cost of Internet services), among others.

With blockchain network infrastructure [11, 46, 12], value transfer can be modeled as a distributed, fail-safe service with minimal cost.

Let a process $\mathcal{P}$ be a miniature money transfer service that meets Bob's purpose. Bob's account balance at a time t can be seen as a state S_t being stored in a blockchain. To transfer a value of x to Alice's account, Bob creates a transaction message $T : \langle t, Bob, Alice, x \rangle$. The transaction T is broadcast over a broadcast network that executes $\mathcal{P}$. Upon receipt of T, certain nodes in the network work on to verify if the transaction is a 'well-formed.' A well-formed transaction must satisfy transaction validity conditions like: "It must be digitally signed by Bob," and "$x <= (\sum_{Bob} + c)$ must be true." When all or majority nodes reach a consensus upon the validity of T, the transaction is stored in a decentralized, publicly verifiable ledger B to complete the transaction. Upon completion of T, $\mathcal{P}$'s state progresses from current state S_t to a globally accepted next state S_{t+1}. More generally, a blockchain network for a process P comprises of m well-defined functions $f_1()...f_n()$; and a universally accepted initial state S_0 stored in an immutable, publicly verifiable ledger B. Execution of a function f_i is triggered by a digitally signed transaction T_i^j, which is broadcast over a broadcast network at time t_i^j. The network nodes interact with each other before reaching a consensus on the validity of the transaction to commit a transaction. A successful execution of function $f_i(T_i^j)$ causes the progression of state from S_t to S_{t+1}. Figure 2.7 shows the transition of system states triggered by transactions in a blockchain network.

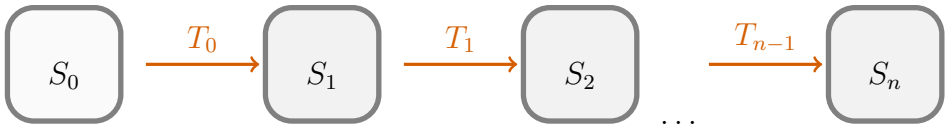

Figure 2.7

A process executed in a blockchain network is a sequence of state transitions. Each state transition (e.g, $S_{n-1} \rightarrow S_n$) is initiated by an atomic operation called transaction (e.g, T_{n-1}). A process, thus, in a blockchain network can be seen as a sequence of committed transactions $T_0, T_1, .., T_{n-1}, T_n$

A blockchain network can be regarded as a universal, trusted platform [11]. The integrity of a process $P(f_0, f_1..f_n)$ executed in the network is rooted in two assumptions: i) immutability of ledger entries, and ii) globally verifiable ledger. Because of a consensus protocol, only globally accepted transaction entries are entered into a blockchain ledger. Even if an invalid entry is made, it can be detected by some of the participating network nodes. Thus, an information system in a blockchain network can always be regarded as secure and trustworthy.

2.5.3 Consensus Protocols

Consensus protocol is a communication protocol by which a network selects a non-faulty or non-malignant node to propose an output of a valid state initiated by an input transaction [47]. Other network nodes verify the output to append it into their local ledger. As shown in Figure 2.9, a node N_1 executes a state transition function $f_i(T_j)$; it purposes a value 40 for the transaction T_j. The peer nodes on the network, N_2, N_3, and N_4 verify the proposed output value; and if determined

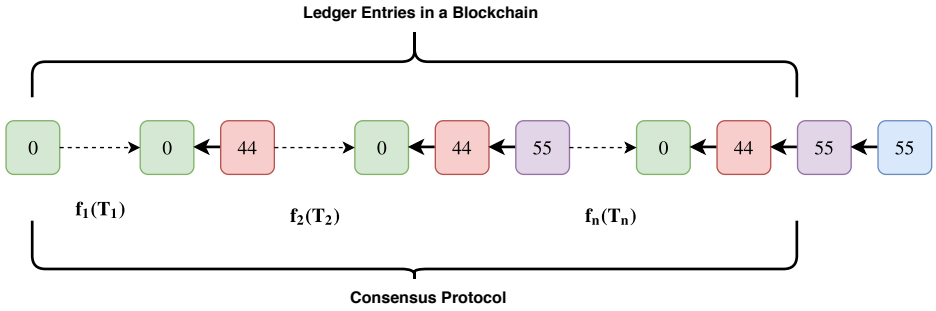

Figure 2.8

A blockchain network uses state transition functions to trigger the progression of the system state.

valid, the output is stored to their local ledger B. Thus, the network always stores a globally agreed-upon state of the system.

In general, a blockchain network involves two types of computing users (nodes) [47]: 1) Incentivised users and 2) Passive users. Incentivised users are also commonly known as miners in the current blockchain networks such as Bitcoin and Etherium. They listen to the broadcast network for incoming transactions. Once they process the received transactions and make a motion about an output of the transactions. Other incentivized users can either accept or reject a motion depending upon their own transaction output. In case multiple incentivized make different motions about the final state of the system, a fork in block-chain occurs. Lightweight computing nodes are the passive users of the blockchain network. They do not always participate in the network but can check sporadically about the integrity of a transaction in times of conflicting motions from different incentivized nodes.

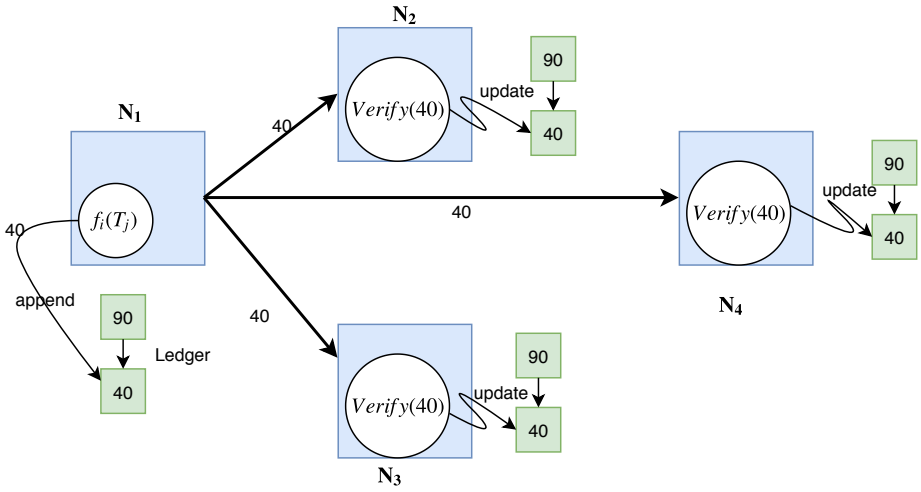

Figure 2.9

Blockchain Network uses consensus algorithms to create an immutable ledger called blockchain or state log.

2.5.3.1 Proof of Work (PoW)

Under this protocol, miners in a blockchain network compete to solve a cryptographic puzzle before one proposes a valid transaction(s) [46, 12, 58, 47, 11]. The first miner to validate the transaction and solve a cryptographic puzzle gets a chance to append the block to its local ledger. Other miners in the network sync their local ledger to that of the first miner if they find the solution to the puzzle is correct for the valid transaction(s). Forking of a blockchain can occur if miners comes with two conflicting correct solutions. For example, miners M_1 and M_2 distant apart in a network can independently solve a puzzle and verify a transaction block. In this case, both miners append to local ledger their own version of the transactions causing a

fork in the blockchain. On solving a cryptographic puzzle, miners expend certain computational power, which renders this protocol as an energy-inefficient protocol.

2.5.3.2 Proof of Stake (PoS)

In PoS protocol, each miner initially stake some amount of value incommensurate to the value of a transaction. A group of such miners are chosen to validate a transaction, which will be finally appended to a blockchain ledger. Upon making a correct motion about the correctness of a transaction and the miners are rewarded value in proportion to their stakes [47, 11]. However, if incorrect motion is made, the miner loses the stake as a penalty.

2.5.4 Explicit and Implicit Process States and Forking

Every blockchain transaction can be seen as a well-defined step in the execution of a process, resulting in a change in the process *state*. For example, a transaction to transfer an amount x from a wallet A to a wallet B results in a change of remaining balances in both wallets. In some blockchain networks like Bitcoin [46], process states are *not* made explicit (ledger entries are merely transactions). Nevertheless, given the complete ledger, anyone can run through all transactions to determine the current state – viz., the remaining balance in every Bitcoin wallet. On the other hand, in the Etherium blockchain network, process (smart-contract) states are also made explicit. Specifically, smart-contract data are captured as a set of key-value pairs. A succinct cryptographic commitment to all key-value pairs, following every transaction, is explicitly specified in every ledger entry. The main advantage of explicit states is that

they act as convenient checkpoints after every transaction. Consider a scenario where there is universal consensus on state ξ_n after the n^{th} transaction T_n; assume that the $(n+1)^{th}$ transaction T_{n+1} results in conflicting motions (like $\xi^1_{n+1}, \xi^2_{n+1}, \ldots$, etc.,) from different incentivized users. To resolve the correct fork, regular users merely have to determine, that given consensus on ξ_n, i) *if* T_{n+1} is well-formed, and if so, ii) How does T_{n+1} affect the state ξ_n (to change state to ξ_{n+1}). The easier it is for users to verify the correctness of any transaction selectively, the riskier it is for incentivized users to make (deliberate or otherwise) incorrect motions.

2.5.5 A Hardware TCB versus Blockchain Network

The information system security model investigated in this dissertation uses Blockchain Network as a chief trustworthy process execution model. However, it is desired to note the rationale of opting out TCB. Trusted Computing Base is by design a hardware intensive solution. Ideally, TCB design require tamper-proof and read/write proof hardware. These requirements are challenging to realize and can cause higher costs. In addition, a TCB by design has low complexity hardware and software, which makes it almost impossible to handle higher throughput and performance. On the other hand, a blockchain network can be explained as a Null TCB computing platform with scalable performance and reliability. Any computing nodes in a blockchain network can join and participate in executing and verifying process outputs. The design principle of a blockchain network is that the integrity of the process is guaranteed because at least some computing node behave honestly. Nonetheless, unlike

TCB, as a ledger is publicly shared by computing nodes, the issue of confidentiality is not addressed by a blockchain network [47]. Nevertheless, several models of private blockchain networks can provide the requirement of privacy. The most significant merit of a blockchain network is the distribution of data and processes. It benefits by evading the problem of a single point of failure, and data centralization [47]. In the following Chapter, we discuss a trustworthy method for identifying simple polygons.

CHAPTER 3

TRUSTWORTHY EXECUTION OF SHAMOS-HOEY ALGORITHM FOR DETECTING A SIMPLE POLYGON

> We can only see a short distance ahead, but we can see plenty there that needs to be done..
>
> *Alan Turing, 1950*

3.1 Introduction

A polygon is a closed figure bounded by a series of boundary line segments on a two-dimensional (2D) plane. A polygon is non-intersecting (or simple) if no two non-adjacent boundary segments on it intersect each other. Several algorithms exist for detecting a simple polygon. For a polygon with N segments (sides/line-segments), a naive algorithm works by testing at most $N * (N - 1)$ unique pairs for intersection. Among other efficient algorithms, two different algorithms by Shamos and Hoey [62], and Bentley and Ottomon [7] are popular ones with a reasonable time complexity of $O(NlogN)$.

Given a 2D polygon, a trustworthy execution of simplified Shamo-Hoey algorithm is a protocol which safely attest if a sequence of connecting vertices form a non-intersecting polygon. A trustworthy execution of an algorithm is intended to

minimize a Trusted Computing Base (TCB) for the algorithm. Minimizing a TCB means that every operation in it can be verified with minimal size(number) proof objects. At the termination, a trustworthy execution of Shamos-Hoey certifies if a given sequence of points (segments) constructs a simple polygon; however, it does not report intersection point(s).

3.2 Shamos-Hoey Algorithm

Shamos-Hoey algorithm is a sweep-line algorithm that processes a queue of lexicographically ordered (by x-values, and y-values) vertices of a polygon. At every left vertex $(x_i, y_i)_l$, the corresponding segment (current segment) is inserted in a self-sorting structure called an active-event tree. The active-event tree maintains the sorted order of segments by the y-value of the intersection of the existing segments and vertical line through the x_i. A segment that is just above the current segment; and another segment just below the current segment from the active-event tree are determined. If either of the segment intersects with current segment, then the algorithm reports detection of intersection. At every right vertex $(x_i, y_i)_r$, the corresponding segment (current segment) is looked up in the active-segment tree to find segments S_A and S_B just above and below the current segment. An intersection between S_A and S_B is checked as if an intersection is found. The algorithm returns immediately; otherwise, the process is repeated for the next vertex in the queue.

Following outlines a simplified generic Shamos-Hoey algorithm [62] for detecting a simple polygon.

Algorithm 1: Shamos and Hoey Algorithm [62]

Input : Set of vertices $\mathcal{S} = \{S_0, S_1..S_N\}$ in a N sided-polygon; S_i connects left vertex $P_i(x, y)$ and right vertex $P_{i+1}(x, y)$

Output: TRUE if any non-adjacent segment $S_i(P_i, P_{i+1})$ and $S_k(P_k, P_{k+1})$ intersects

```
1  ξ = {e_i←Event(S_i · P_i), e_j←Event(S_i · P_{i+1})} ;   // Event object e_i
     stores P_i, and segment S_i
2  ξ = SORT(ξ) ;   // Sort by event's P_i · x-values and P_i · y-values.
3  T = {} ; // T is self height-balancing binary tree of Segments in
     S, sorted by key:S · y
4  for i:=1 UNTIL 2N do
5      P := ξ[i] · P ;                    // vertex for this event ξ[i]
6      S := ξ[i] · S ;                    // Segment for this event ξ[i]
7      if P is the Left Vertex of S then
8          INSERT(S, T) ;                          // insert S into T
9          A:=Above(S, T) ;          // A, Segment just above S in T
10         B:=Below(S,T) ;           // B, Segment just below S in T
11         if Intersect(S,A) then
12             RETURN TRUE
13         end
14         if Intersect(S,B) then
15             RETURN TRUE
16         end
17     else
18         if P is the Right Vertex of S then
19             A:=Above(S, T);
20             B:=Below(S,T);
21             if Intersect(A,B) then
22                 RETURN TRUE
23             end
24             DELETE(S, T) ;          // Removes segment S from T
25         end
26     end
27 end
28 RETURN FALSE
```

In this section, we use the following set of symbols and operations to discuss a a trustworthy execution of a simplified Shamos-Hoey algorithm.

- Operator ':=' is for a variable assignment, '=' compares two variables, '←' is sets a value to a variable.
- $h \leftarrow H(v)$ is a collision-resistant hash function that takes an input value v to produce a fixed-size digest string h.

- $UDIs$ is an acronym for Unconstrained Data Items, which is a set of data input to a process. They are part of a transaction that trigger state transitions of a process. Preconditions are a set of statements that must be verified against UDIs of a transaction before executing a transaction or a procedure. Post-conditions are the set of operations that must be executed for a procedure to fix a state change of a system triggered by a transaction.

In the following section, we discuss operations of a trustworthy execution of the simplified Shamos-Hoey algorithm.

3.3 Trustworthy Execution of Simplified Shamos-Hoey Algorithm

Let P be a polygon bounded by a counter-clockwise ordered segments, $S = \{s_0, s_1, s_2, ..., s_n\}$. Each segment $s_i \in S$ is a geometry in 2-dimensional plane connecting two distinct points $A(x_1, y_1)$, and $B(x_2, y_2)$ such that A is left endpoint and B is the right endpoint satisfying if $x_1 \neq x_2, x_1 < x_2$ and if $x_1 = x_2, y_1 < y_2$. A segments s connecting two points (x_1, y_1) and (x_2, y_2), and other segment s' connecting two distinct points (x'_1, y'_1) and (x'_2, y'_2) can be expressed using a parametric equation as:

$$L_s = \begin{pmatrix} x_1 \\ \\ y_1 \end{pmatrix} + t \begin{pmatrix} x_2 - x_1 \\ \\ y_2 - y_1 \end{pmatrix}, \; L'_s = \begin{pmatrix} x'_1 \\ \\ y'_1 \end{pmatrix} + u \begin{pmatrix} x'_2 - x'_1 \\ \\ y'_2 - y'_1 \end{pmatrix} \tag{3.1}$$

Solving equations L_s, and L'_s we get scalar values t and u as:

$$\begin{aligned} t &= \frac{(x_1 - x'_1)(y'_1 - y'_2) - (y_1 - y'_1)(x'_1 - x'_2)}{(x_1 - x_2)(y'_1 - y'_2) - (y_1 - y_2)(x'_1 - x'_2)} \\ u &= -\frac{(x_1 - x_2)(y_1 - y'_1) - (y_1 - y_2)(x_1 - x'_1)}{(x_1 - x_2)(y'_1 - y'_2) - (y_1 - y_2)(x'_1 - x'_2)} \end{aligned} \tag{3.2}$$

s and s' intersects on the lines between two points if $0 \leq t \leq 1.0$ and $0 \leq u \leq 1.0$. A merit of this formulation is that the conditions can be tested without actually dividing the numerator by the denominator, which is a determinant of four points. Thus permits faster determination of existence of any intersection before computing actual intersection point [23].

A trustworthy execution of a simplified Shamos-Hoey algorithm is a protocol stack of three major sub-processes, which are executed in sequence following stricter preconditions and postconditions.

1. The first sub-process $\mathcal{P}_1$ takes a series of segments in a 2D polygon $P = \{s_0, s_1, ...s_n\}$ to construct a special structure called segment tree $\mathcal{T}_S$, which is an OMT with a static (computed only once) root σ_S. Each segment connecting two endpoints (x_1, y_1) and (x_2, y_3) is stored as 3-tuple (k, k_n, v) leaf node in the tree. The key k is the segment's positional index (ID) when segments are ordered in a counter-clockwise (CCW); and value v is the segment's endpoint (x_1, y_1, x_2, y_2); and k_n is the index of succeeding segment. The rationale behind building an OMT for the input segments is that the leaf nodes are automatically ordered by node's key value k. It is also useful to look-up entry with the lowest and the highest key among the leaves of an OMT.

2. The second sub-process $\mathcal{P}_2$ takes a series of segments from a segment tree $\mathcal{T}_S$ to construct an event tree $\mathcal{T}_E$, which is an OMT with a dynamic Merkle root σ_E. To each segment in $\mathcal{T}_S$, the left endpoint (x_1, y_1) is inserted as an ESTART event leaf node of the tree; and right endpoint (x_2, y_2) inserted as an ESTOP event leaf node of the tree. Each ESTART leaf node is a 3-tuple entry (α, α_n, v), where α is x-value of an endpoint (x_1, y_1); and v is 5-tuple $(x_1, y_1, x_2, y_2, \mathsf{ESTART})$ containing segment's endpoints, and event type ESTART; α_n is the next higher event entry in the tree. Similarly, an ESTOP leaf node is also stored as (α, α_n, v) for a right endpoint (x_2, y_2); where $\alpha = x_2$, and $v = (x_1, y_1, x_2, y_2, \mathsf{ESTOP})$. All leaf nodes in an event tree together is a collection of vertices in the polygon extracted out of a segment tree $\mathcal{T}_S$. The vertices are lexicographically ordered by x-ordinates from left to right on the plane. This tree also supports query for a leaf node entry with the lowest key and the highest key.

3. The third sub-process $\mathcal{P}_3$ extract events from an event-tree $\mathcal{T}_E$ in the order of their occurrence, to construct an active segment tree T_A, which is an OMT with a dynamic Merkle root σ_E. If an input event $e : (\alpha, \alpha_n, v =$

$(x_1, y_1, x_2, y_2, \{\mathsf{ESTART}, \mathsf{ESTOP}\}))$ is of type ESTART; this event is entered as an 'active-event' leaf node of $\mathcal{T}_A$. Each leaf node is a 3-tuple (γ, γ_n, v) entry, where γ is y-value (y_1) of the left endpoint of the segment in the event e; γ_n is the next higher leaf node in this tree. If an input event e is of type ESTART; a segment corresponding to this event is looked up into the active-event tree $\mathcal{T}_A$ to determine the intersection between segments.

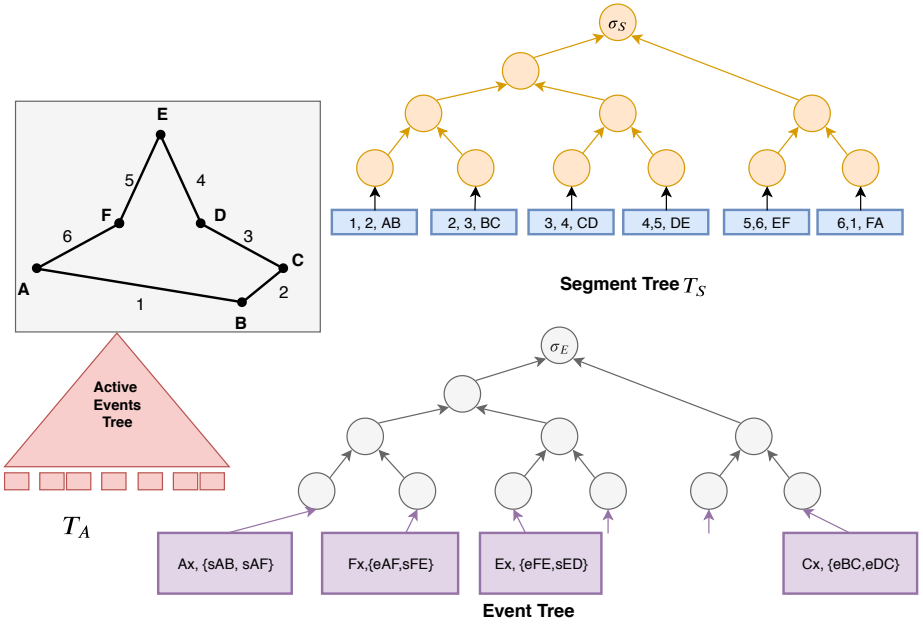

Figure 3.1

OMT structures in a trustworthy execution of the simplified Shamos-Hoey Algorithm.

Figure 3.1 shows simplified structures that support a trustworthy execution of the Shamos-Hoey algorithm for attesting (authorizing) a simple polygon. For simplicity, we assume following cases regarding vertices of a given polygon P $\{s_0, s_1, s_2, ..., s_n\}$:

1. Segments $s_0, s_1..s_n$ are in counter-clockwise ordering in the polygon.
2. No two vertices in vertex sequence $v_0, v_1..v_n$ of the polygon share common X and Y projection. We can always relax this restriction by protruding aligned

vertex by infinitely small change in both X and Y to produce uniqueness in X and Y-ordinates.

In what follows, we describe each of the sub-processes $\mathcal{P}_1..\mathcal{P}_3$ in details:

3.3.1 Sub-Process $\mathcal{P}_1$

The first sub-process starts with an empty OMT known as a segment tree $\mathcal{T}_S$, with initial root value σ_S. The process $\mathcal{P}_1$ executes on by taking a counter-clockwise ordered segments $S = \{s_0, s_1, ..s_n\}$ of an input polygon P with n-sides. Each of the input segment $s_i \in S$ is processed in that order to construct a segment tree $\mathcal{T}_S$, whose leaf nodes store information about the endpoints of the segment. Let an i^{th} input segment $s_i \in S$ connects two distinct endpoints (x_1, y_1) and $x_2, y_2)$. It is added (inserted) as a leaf node in the segment tree $\mathcal{T}_S$. Each leaf node in the tree is a 3-tuple entry of the form:

$$(k := i, k_n : j, v := (x_1, y_1, x_2, y_2))$$

where, j is the index of the next higher order segment. An OMT insertion protocol is followed to insert an item into the segment tree. After each insertion, the current root $\sigma_S \in \mathcal{T}_S$ is updated to reflect a change in the tree. After inserting n^{th} segment in $P = \{s_0, s_1, ...s_i...s_n\}$; the leaf nodes (entries) of the segment tree is a collection:

$$\{(k := 0, k_n : 1, v := (x_1, y_1, x_2, y_2)_{s_0}), ...(k := n, k_n : 0, v := (x_1, y_1, x_2, y_2)_{s_n})\}$$

The root of value is σ_S is a single commitment to all leaf nodes in the tree, which are sides(segments) of an input polygon object. The utility of a segment tree $\mathcal{T}_S$ is that its root value σ_S is signed and published by an untrusted source/directory; and a prover can verify membership of a segment in an input segments $S = \{s_0, s_1, ...s_n\}$. All that is needed to verify the existence of a leaf L_i is a verification object $\mathcal{VO}_i$, which is a set of complementary nodes (hashes) for the leaf node. The protocol for such verification is explained in section 2.4.5.

3.3.2 Sub-Process $\mathcal{P}_2$

The sub-process $\mathcal{P}_2$ starts after completion of $\mathcal{P}_1$ that constructs a segment tree $\mathcal{T}_S$. The process bootstraps by initializing an empty OMT tree called event tree $\mathcal{T}_E$, with an initial root value of σ_E. Inputs to process $\mathcal{P}_2$ are the segments from an ordered sequence segments in a segment tree $\mathcal{T}_S$ and a signed root σ_S built by process $\mathcal{P}_1$. It processes leaf nodes in a segment tree one by one, beginning from lowest key entry; and expecting next higher key entry in the segment tree. Upon receipt of an i^{th} segment entry $(i, j, v := (x_1, y_1, x_2, y_2))_{s_i}$ corresponding to a segment s_i, the process $\mathcal{P}_2$ executes AddEvent procedure to insert two leaf nodes into the event tree $\mathcal{T}_E$. The left endpoint $A(x_1, y_1)$ is inserted as a start event leaf node, ESTART; while the right endpoint $B(x_2, y_2)$ is inserted as an ESTOP event leaf node. More specifically,

for each left endpoint $A(x_1, y_1)$ of a segment s_i, a leaf node/entry L_i^E represents an START event in tree $\mathcal{T}_E$ is defined as:

$$(\alpha := x_1, \alpha_n, v := \{x_1, y_1, x_2, y_2, \mathsf{ESTART}\}) \tag{3.3}$$

,where, the key is $\alpha = x_1$; α_n is the next higher key to this leaf; value v is a dictionary containing segment's endpoints and event type ESTART. Similarly, for the right endpoint $B(x_2, y_2)$ of the segment s_i, a leaf structure L_i^E represents a STOP event in $\mathcal{T}_E$ is defined as:

$$(\alpha := x_2, \alpha_n, v := \{x_1, y_1, x_2, y_2, \mathsf{ESTOP}\}) \tag{3.4}$$

, where key is $\alpha = x_2$; α_n is the next higher key to this leaf; value v is a dictionary containing segment's endpoints and event type ESTOP.

Insertion of a new endpoint $A(x', y')$ for a new segment s' is performed as follows:

- if a leaf node L^E with key $\alpha = x'$ exists in $\mathcal{T}_E$, then value v is updated by 5-tuple t:

$$t = \begin{cases} (x', y', x_2, y_2, \mathsf{ESTART}), & \text{if } A(x', y') \text{ is left endpoint of } s' \\ (x_1, y_1, x', y', \mathsf{ESTOP}) & \text{otherwise} \end{cases}$$

- otherwise, new entry is inserted is:

$$L_{s'}^E = \begin{cases} (\alpha = x', \alpha_n, v := \{x', y', x_2, y_2, \mathsf{ESTART}\}), & \text{if } A(x', y') \text{ is left endpoint of} s' \\ (\alpha = x', \alpha_n, v := \{x_1, y_1, x', y', \mathsf{ESTOP}\}) & \text{otherwise} \end{cases}$$

Every insertion or update of the items in an event tree $\mathcal{T}_E$ causes an update of the root value σ_E to reflect the change. The process can be algorithmically expressed as in Table 3.3.2.

Table 3.1

Formal description of the trustworthy execution of the modified Shamos-Hoey Algorithm with preconditions and postconditions for the sub-process P_2.

Operation AddEvent

// endpoints of a segment are added as events to an event tree T_E.

UDIs:

$(x_i, y_i, x_j, y_j, T_S/\hat{\sigma_S}, T_E/\sigma_E)$

//endpoints of a segment s_i, reference to T_S, and T_E;

//this module will find the EventType for each endpoint

Preconditions:

ESTART $\leftarrow 0$; ESTOP $\leftarrow 1$

$k_s \leftarrow H(x_i, y_i, x_j, y_j); v_s \leftarrow (x_i, y_i, x_j, y_j);$ // compute index/key, value

$(k_s, v_s) \in T_S/\hat{\sigma_S}$ //segment exists in a segment tree

// left endpoint @x_i–right endpoint @x_j

$x_i < x_j : EventType1 \leftarrow$ ESTART

if$(x_i = x_j$ and $y_i < y_j) : EventType1 \leftarrow$ ESTART

else :RETURN

$EventType2 \leftarrow$ ESTOP

$ek_i \leftarrow x_i; ev_i \leftarrow (x_i \cdot y_i \cdot x_j \cdot y_j \cdot EventType1);$ // first event

$[ek_i, ev_i] \notin T_E/\sigma_E$ //event does not exist in event tree

$ek_j \leftarrow x_j; ev_j \leftarrow (x_i \cdot y_i \cdot x_j \cdot y_j \cdot EventType2);$ // second event

$[ek_j, ev_j] \notin T_E/\sigma_E$ //event does not exist in event tree.

Postconditions:

$T_E/\sigma_E \leftarrow$ InsertLeaf$(\{ek_i, ev_i\});$ // new leaf node into event tree, and update its root.

$T_E/\sigma_E \leftarrow$ InsertLeaf$(\{ek_j, ev_j\});$

//delete a segment from segment tree and update MT root.

$T_S/\hat{\sigma_S} \leftarrow$ DelLeaf$(\{k_s, v_s \leftarrow 0\});$

Preconditions for inserting a segment $s_i : A(x_i, y_i) - B(x_j, y_j)$ into an event OMT $\mathcal{T}_E$ is that the segment-

i) exist in the segment tree $\mathcal{T}_S$. Given a segment's endpoints as UDIs to this process, it is implicit that verification objects $\mathcal{VO}$ for the existence of the segment in $\mathcal{T}_S$ is passed along with the UDIs, and

ii) does not exist in the event tree T_E. It is also in implicit that verification objects $\mathcal{VO}$ for the non-existence in $\mathcal{T}_E$ is passed along with the UDIs.

Once the preconditions are verified, events are inserted in an event tree T_E; and ultimately, the segment is deleted from segment tree T_S. The Merkle roots of both OMT trees are updated to reflect the insertion and deletion of leaf nodes. This process terminates by ensuring all segments in T_S have been added as events in the event tree $\mathcal{T}_E$, leading segment tree $\mathcal{T}_S$ to be empty. On termination, this process constructs an OMT tree whose leaf nodes represent the sorted events for an array of endpoints of the segments in an input polygon. Listed below are interfaces of two simple operations invoked.

- AddEvent$(T_E, \sigma_E, x_i, y_i, x_j, y_j)$; is an operation that adds two new events into an event tree $\mathcal{T}_E$, with current root σ_E.
- DelEvent$(\alpha : x', v' = (x_1, y_1, x_2, y_2, \{\mathsf{ESTART}, \mathsf{ESTOP}\}))$; is an operation that updates value v of an existing event entry $L^E \in \mathcal{T}_E$ defined as:

  $$L^E : (\alpha = x', \alpha_n, v = (x_1, y_1, x_2, y_2, \{\mathsf{ESTART}, \mathsf{ESTOP}\}))$$

  such that if $v' = v$, $v \to 0$.

3.3.3 Sub-Process $\mathcal{P}_3$

The third sub-process $\mathcal{P}_3$ starts after completion of $\mathcal{P}_2$. It bootstraps by initializing an empty OMT called an active event tree $\mathcal{T}^A$, with an initial root σ_A. It

processes events L_i^E in order starting from the lowest key event leaf L_0^E and expects next higher event leaf in event tree $\mathcal{T}_E$. Each of the event leaf $L_i^E \in \mathcal{T}_E$, as expressed by (3.3), is an entry

$$(\alpha_0 = x_1, \alpha_n, v = (x_1, y_1, x_2, y_2, \{\mathsf{ESTART}, \mathsf{ESTOP}\}, x_1', y_1', x_2', y_2', \{\mathsf{ESTART}, \mathsf{ESTOP}\}))$$

where, the first 5-tuple $(x_1, y_1, x_2, y_2, \{\mathsf{ESTART}, \mathsf{ESTOP}\})$ represents an event for a segment s and second 5-tuple $(x_1', y_1', x_2', y_2', \{\mathsf{ESTART}, \mathsf{ESTOP}\})$ represents an event for second segment s' where s and s' are adjacent segments in a polygon sharing of one the vertices (x_1, y_1).

Thus this process operates on two events encoded into a node of an event tree. The first type of event is an ESTART event and the second type is an ESTOP event.

If L_i^E is an event type START for a segment s_i, the following operations are performed:

- insert a leaf L_i^A for the segment $s_i : (x_1, y_1, x_2, y_2) \in L_i^E$ into an active event OMT $\mathcal{T}_A$. A leaf entry L_i^A is defined as:

  $$(\gamma : y_1, \gamma_n, v := (x_1, y_1, x_2, y_2)),$$

  where, γ_n is the next higher key for an event.

- search in T^A a leaf L_{bel}^A just below current leaf L_i^A. L_{bel}^A contains a segment s_{bel}, which falls just below the segment s_i;

- search in T^A a leaf L_{abv}^A just above current leaf L_i^A. L_{abv}^A contains a segment s_{abv}, which is just below the segment s_i.

- return True if segment s_i intersects with either s_i or s_{abv}); otherwise, accept next higher event leaf

If L_i^E is an event type ESTOP that contains segment s_i, the following operations are performed:

- search in T^A a leaf L_{bel}^A just below current leaf L_i^A. L_{bel}^A contains a segment s_{bel} , which falls just below the segment s_i;
- search in T^A a leaf L_{abv}^A just above current leaf L_i^A. L_{abv}^A contains a segment s_{abv} which is just below the segment s_i.
- return True if segment s_{abv} intersects with segment s_{bel}.
- delete leaf L_i^A containing segment $s_i : (x_1, y_1, x_2, y_2) \in L_i^E$ into an active event OMT $\mathcal{T}_A$.

Algorithmically this process is explained in Table 3.3.3.

The first pre-condition to bootstrap this process $\mathcal{P}_3$ is an event with minimum key in $\mathcal{T}_E$. This event must be verified against the minimum key in $\mathcal{T}_E$. It keeps track of the next event expected in a register $'rmin'$ set to it by the next key index (α_n) of a current event. Register $rmax$ is used to keep track of the last event leaf expected when the process will terminate.

The other two preconditions verify that s_{abv} is above the current segment $s_i : (x_i, y_i, x_j, y_j)$, and s_{bel} is below segment $s_i : (x_i, y_i, x_j, y_j)$ of an input event. It is implicit that verification objects $\mathcal{VO}$ for these verifications are supplied along with UDIs. Listed below are the operation definitions of required operations.

- $s_{abv} \leftarrow$ AboveActSeg(x_i, y_i, x_j, y_j); returns an active segment just above the current segment whose leaf index is y_i in T_A. In case no segment exists above the given segment, a Null is returned.
- $s_{bel} \leftarrow$ BelActSeg(x_i, y_i, x_j, y_j); returns an active segment just below the current segment whose index is y_i in collection whose OMT is $\mathcal{T}_A$. In case no segment exists above the given segment, a Null is returned.
- $\{TRUE, FALSE\} \leftarrow$ VerifyAbvSeg$(s_i : x_i, y_i, x_j, y_j, s_{abv} = \{(x'_i, y'_i, x'_j, y'_j), Null\})$ returns TRUE if current seg s_{cur} in $\mathcal{T}_A$ is above the given segment s_{abv} in T_A.

Table 3.2

Formal description of the trustworthy execution of the modified Shamos-Hoey Algorithm with preconditions and postconditions for the sub-process P_3

Operation: ProcessEvent // Process each event in the Event Tree in order.
UDIs:
$(\mathsf{x_i}, \mathsf{y_i}, \mathsf{x_j}, \mathsf{y_j}, \mathsf{EventType}, \sigma_\mathsf{E}, \mathsf{T_E})$// current segment from event tree T_E
$(\mathsf{x_i'}, \mathsf{y_i'}, \mathsf{x_j'}, \mathsf{y_j'}, \mathsf{T_A})$ //segment below current segment in active event tree T_A
$(\mathsf{x_i''}, \mathsf{y_i''}, \mathsf{x_j''}, \mathsf{y_j''}, \mathsf{T_A})$/ segment above current segment in active event tree T_A

Preconditions:

// initialize registers
$\mathsf{ESTART} \leftarrow 0; \mathsf{ESTOP} \leftarrow 1; \mathsf{rmin} \leftarrow 0; \mathsf{rmax} \leftarrow 0$
$ek_i \leftarrow x_i; v_i \leftarrow H(x_i \cdot y_i \cdot x_j \cdot y_j \cdot EventType)$
$\mathsf{curSeg} \leftarrow (\mathsf{x_i}, \mathsf{y_i}, \mathsf{x_j}, \mathsf{y_j})$
$\mathsf{rmin} = 0$:
    $\mathsf{rmin} \leftarrow \mathsf{OmtMinKey}(\mathsf{T_E})$;
    $\mathsf{rmax} \leftarrow \mathsf{OmtMaxKey}(\mathsf{T_E})$
ELSE : CONTINUTE
$\mathsf{rmin} = \mathsf{ek_i}$:
    $\mathsf{rmin} \leftarrow \mathsf{NextKey}(\mathsf{ek_i})$;
ELSE : RETURN
$[ek_i, v_i] \in T_E/\sigma_S$ //event exists in event tree
$T_S/\acute{\sigma_S} = 0$ //segment tree root value is 0
$\mathsf{segAbv} \leftarrow \mathsf{AboveActSeg}(\mathsf{T_A}, \mathsf{x_i}, \mathsf{y_i}, \mathsf{x_j}, \mathsf{y_j})$
$\mathsf{segBel} \leftarrow \mathsf{BelActSeg}(\mathsf{T_A}, \mathsf{x_i}, \mathsf{y_i}, \mathsf{x_j}, \mathsf{y_j})$
$\mathsf{VerifyAbvSeg}(\mathsf{s_{cur}}, \mathsf{segAbv})$
$\mathsf{VerifyBelSeg}(\mathsf{s_{cur}}, \mathsf{segBel})$

Postconditions:

$\mathsf{EventType} = \mathsf{ESTART}$:
  $ak_i = y_i$
  $\mathsf{AddActSeg}(\mathsf{T_A}, \mathsf{x_i}, \mathsf{y_i}, \mathsf{x_j}, \mathsf{y_j})$
  $\mathsf{rmin} \leftarrow \mathsf{OmtMinKey}(\mathsf{T_E})$;
  $\mathsf{tmp} \leftarrow \mathsf{Intersect}(\mathsf{curSeg}, \mathsf{segAbv})$;
  $\mathsf{tmp} = \mathsf{TRUE}$:
    RETURN tmp
  $\mathsf{tmp} \leftarrow \mathsf{Intersect}(\mathsf{curSeg}, \mathsf{segBel})$;
  $\mathsf{tmp} = \mathsf{TRUE}$:
    RETURN tmp
ELSE :
  $\mathsf{T_A}/\sigma_\mathsf{A} \leftarrow \mathsf{DelActSeg}([\mathsf{ak_u}, \mathsf{v_i}])$
  $\mathsf{RETURN} \leftarrow \mathsf{Intersect}(\mathsf{segAbv}, \mathsf{segBel})$;
  $\mathsf{tmp} = \mathsf{TRUE}$:
    RETURN tmp
$\mathsf{rmin} \leftarrow \mathsf{OmtMinKey}(\mathsf{T_E})$; //Update register
$\mathsf{T_E}/\sigma_\mathsf{E} \leftarrow \mathsf{DeleteEvent}(\mathsf{ek_i})$

If s_{abv} is Null, then verification leading to the non-existence of such a segment must be performed.

Let s_{cur} be a denoted by a leaf node $[key_{cur}, key_{cur-next}, s_i] \in \mathcal{T}_{\mathcal{A}}$ Let s_{abv} be a denoted by a leaf node $[key_{abv}, key_{abv-next}, s_{abv}] \in T_A$. if $key_{cur-next} = key_{abv-next}$ then, the s_{abv} is above s_i.

- $\{TRUE, FALSE\} \leftarrow$ VERIFYBELSEG($s_i : x_i, y_i, x_j, y_j, s_{bel} : x'_i, y'_i, x'_j, y'_j$) returns TRUE if current segment s_i in T_A is below the given segment s_{bel} in $\mathcal{T}_A$. If s_{bel} is Null, then verification leading to the non-existence of such a segment must be performed.

- ADDACTSEG($\mathcal{T}_{\mathcal{A}}, x_i, y_i, x_j, y_j$); is an operation that adds a new event in an event collection C_A whose OMT is T_A.

- DELACTSEG($index : x_i, x_i, y_i, x_j, y_j, EventType$); is an operation that removes an existing active segment in an active segment tree $\mathcal{T}_A$.

- $\{TRUE, FALSE\} \leftarrow$ VERIFYNONADJACENT($seg_i : (,,,), seg_j : (,,,)$) returns True:if seg_i and seg_j are not adjoining segments of a polygon P.

There are two major postconditions in this sub-process that must be verified.

i) If an input event is a start event, it is added as a leaf node into an active segment tree T_A. This event's segment s_i is tested Intersection with two segments in an event tree: 1) segment s_{abv} just above s_i, and 2) segment s_{bel} just below s_i. If an intersection is found, then this process immediately terminates; otherwise, the process expects the new events to process.

ii) If an input event is a stop event, it tests Intersection between s_{abv} and s_{bel} segments. If an intersection exists, then the process exits immediately; otherwise, the process deletes the processed event from the event tree, expecting the next higher event to process.

The algorithmic description of INTERSECTION operation with preconditions and postconditions is listed in Table 3.3. It determines if an intersection exists between to segments s and s' using an efficient method expressed in 3.2.

Table 3.3

Formal description of the trustworthy execution of the modified Shamos-Hoey Algorithm with preconditions and postconditions for Intersection Procedure.

Operation INTERSECTION

// Returns intersection point of two segments s and s'

UDIs:

$\mathsf{s} : (\mathsf{x_i}, \mathsf{y_i}, \mathsf{x_j}, \mathsf{y_j})$// first segment

$\mathsf{s'} : (\mathsf{x'_i}, \mathsf{y'_i}, \mathsf{x'_j}, \mathsf{y'_j})$ // second segment

Preconditions:

$x_i < x_j$ //segment is ordered in increasing x-value

VERIFYNONADJACENT(S,S') //Simple polygions have adjacent segments intersect at an end-point.

Postconditions:

// (x', y') left of seg $(x_i, y_i) - (x_j, y_j)$

// > 0 for left, 0 for on, and < 0 for right of the line

Function : $\mathbf{IsLeft}(\mathbf{x_i}, \mathbf{y_i}, \mathbf{x_j}, \mathbf{y_j}, \mathbf{x'}, \mathbf{y'})$

Return : $\mathsf{isleft} \leftarrow (\mathsf{x_j} - \mathsf{x_i}) * (\mathsf{y'} - \mathsf{y_i}) - (\mathsf{x'} - \mathsf{x_i}) * (\mathsf{y_j} - \mathsf{y_i})$

// Return the intersecion point (x,y)

Function : $\mathbf{IntersectionPoint}(\mathbf{x_i}, \mathbf{y_i}, \mathbf{x_j}, \mathbf{y_j}, \mathbf{x'_i}, \mathbf{y'_i}, \mathbf{x'_j}, \mathbf{y'_j})$

$a1 = y_j - y_i; b1 = x_i - x_j; c1 = a1 * x_i + b1 * y_i$

$a2 = y'_j - y'_i; b2 = x'_i - x'_j; c2 = a1 * x'_i + b1 * y'_i$

$d = a1 * b2 - a2 * b1$

$x = b2 * c1 - b1 * c2$

$y = a1 * c2 - a2 * c1$

Return : (x, y)

$\mathsf{isleft_i} = \mathsf{IsLeft}(\mathsf{x_i}, \mathsf{y_i}, \mathsf{x_j}, \mathsf{y_j}, \mathsf{x'_i}, \mathsf{y'_i})$

$\mathsf{isleft_j} = \mathsf{IsLeft}(\mathsf{x_i}, \mathsf{y_i}, \mathsf{x_j}, \mathsf{y_j}, \mathsf{x'_j}, \mathsf{y'_j})$

$\mathsf{isleft_k} = \mathsf{IsLeft}(\mathsf{x'_i}, \mathsf{y'_i}, \mathsf{x'_j}, \mathsf{y'_j}, \mathsf{x_i}, \mathsf{y_i})$

$\mathsf{isleft_l} = \mathsf{IsLeft}(\mathsf{x'_i}, \mathsf{y'_i}, \mathsf{x'_j}, \mathsf{y'_j}, \mathsf{x_j}, \mathsf{y_j})$

$\mathsf{AssertGt}(\mathsf{isleft_i} \times \mathsf{isleft_j}, 0)$:

Return : FALSE

$\mathsf{AssertGt}(\mathsf{isleft_k} \times \mathsf{isleft_l}, 0)$:

Return : FALSE

RETURN TRUE, $\mathsf{IntersectionPoint}(\mathsf{s}, \mathsf{s'})$

3.4 Related Works, Applications and Conclusion

A trustworthy execution of the Shamos-Hoey algorithm is a subsidiary to trustworthy execution of several secure services based on spatial data assets. For instance, the input polygon to a polygon decomposition algorithm must be a simple geometric structure. Algorithms for a point location problem, convex-hull, Delaunay triangulation also take simple polygon as an input. However, if the integrity of a simple polygon is compromised, the output of the dependant algorithms is highly likely to be compromised, otherwise incorrect.

CHAPTER 4

SECURE QUERYABLE DYNAMIC MAPS

> The economics of the security world are all horribly, horribly nasty and are largely based on fear, intimidation and blackmail
>
> *Linus Torvalds*

4.1 Introduction

As shown in Figure 4.1, consider a set of non-intersecting, closed polygons R_1, R_2, R_3 and R_4 bounded by sequences of connected points

$$\{A, N, M..C, B, A\}, \{A', B', C'..E', A'\}..\{D, E, F'..I', D\}, \{K'L', M', ..O', K'\}$$

The region R_1 is an island polygon inside region R_2, and the region R_4 is completely excluded from other regions R_1, R_2, R_3. Let the region exterior to polygons R_1, R_2, R_3, R_4 be ϕ. For convenience, let the unique codes $\rho_1, \rho_2, ..\rho_4$ be used to represent the region interior to each of the polygons $R_1, R_2..R_4$, respectively. A secure queryable dynamic map (SQDM) for a set of polygonal regions $\{R_0, R_1, ..R_n\}$ is defined as:

$$R_{SQDM} = SQDM(R_0, R_1, ..R_n),$$

where R_{SQDM} is a dynamic structure that supports the following operations that:

- serves authoritative and unbiased query response about the location of a query point $q(x, y)$.
- facilitates incremental trustworthy/verifiable insertion or merging of any subdivision on the map.
- serves AU services pertaining to associating any attribute to a point in a geographic region.

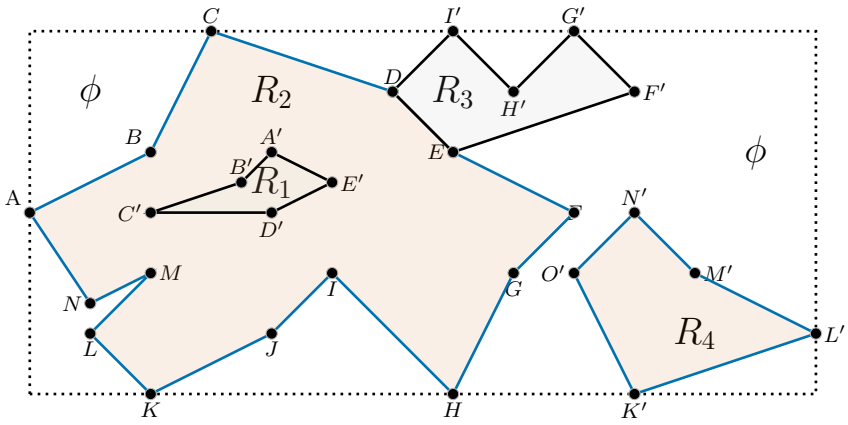

Figure 4.1

A set of polygons on a map. Polygon $ABC..NA$ contains an island polygon $A'E'D'..B'A'$.

In general, the input to an SQDM process $\mathcal{P}$ is a sequence of vertices $S = \{p_0, p_1, ...p_n\}$ corresponding to non-intersecting (non-crossing), closed polygons $R_0, ..R_n$, where each point is a tuple (x, y) that represents a point in 2D. Two consecutive points make a segment that separates two regions in the plane. Execution of the SQDM process $\mathcal{P}$:

1. maps each line to a rectangular block $[x_1, x_2, y_1, y_2, v]$ (or simply bounding box (BB)) (as shown in Figure 4.2 (Right) in a mesh (a set of BBs) constructed for the regions and

2. assigns an appropriate context (say, α) specific value v to all points in the BB, where $v = \{\}$ implies α-specific interpretation of such BBs. For instance, context alpha can be a zip code, street, city, congressional districts, and so on over the BB.

Thereafter, responding to a query regarding any point $q(x, y)$ boils down to a trivial problem of identifying the BB containing $q(x, y)$, and using the value v to answer the query. With this definition, the next section is dedicated to briefly explain a trustworthy execution of $SQDM$ process to build R_{SQDM} that serves the above-mentioned operations. From this now on, we use the symbols p, p', or upper case letters such as A, and B to represent a 2D point. The x and y ordinate of any point p is designated as p_x and p_y (or A_x and A_y); region above a segment AB is designated by $AB.\rho_a$, region code below a segment AB is accessed by $AB.\rho_b$.

4.2 Sub-processes in SQDM Protocol

An SQDM Protocol can be divided into four sub-processes:

1. $\mathcal{P}_0$: to i) pre-process input polygons to create a boundary-point OMT $\mathcal{T}_p$ with dynamic root σ_p to capture the integrity and validity of the input boundaries and ii) create a template mesh, $\mathcal{B}$, which is a set of bounding boxes (BBs) that enclose complete segments in the boundaries. More specifically, a template mesh is a 2D OMT $\mathcal{T}_B$ with dynamic root σ_B called region OMT, whose leaf items are entries of the form $[x_1, x_2, y_1, y_2, v]$.

2. $\mathcal{P}_1$ maps segments in an input polygon to one of the BB according to a specific rule.

3. $\mathcal{P}_2$ utilizes the set of BBs constructed by $\mathcal{P}_1$ to authoritatively resolve queries; and

4. $\mathcal{P}_3$, that allows incremental changes to be performed to the BBs (or to the values in BBs) by making batch execution of $\mathcal{P}_1$ possible. For instance, a BB can be split vertically or horizontally into constituent BBs; two adjoining/neighboring BBs can be merged together; splits mapped to a BB can be split into constituent

splits, new segments can be added to create subdivisions inside an existing region (polygons), among others.

In the blockchain-based SQDM execution, the process states are represented as a 2-tuple (σ_p, σ_B). Every valid operation executed on the input map progresses the current state $S_t : (\sigma_p, \sigma_B)_t$ to new state $S_{t+1} : (\sigma'_p, \sigma'_B)_{t+1}$. Such progress is committed only after verifying the correctness of the new state, which is supported by $O(n)$ verification objects, VOs. In other words, an SQDM in blockchain is bootstrapped by initializing a blockchain state as:

$$S_0 : (\sigma_p \leftarrow 0, \sigma_B \rightarrow)_{t_0}$$

At the crux of a blockchain-based SQDM protocol are micro-transactions that make up each of the processes $\mathcal{P}_0, \mathcal{P}_1 ... \mathcal{P}_3$. The micro- transactions are elementary, individual operations that composed of a larger, complex process. Such transactions are committed for each of the processes $P_1..P_2$ only after verifying specific transaction conditions by an independent, multiple validator (or say verifiers)s in a blockchain network. The way it works ensures the integrity of every data and process outcome from very genesis to a termination of an SQDM process. Confidence in underlying, resultant data structure facilitates an authoritative response to any query against input data. To build such a system, we leverage features of an OMT that allow verifiable, and incremental updates to any data record, SQDM blocks BBs in $\mathcal{B}$; and other context-specific data) in its leaf nodes. In what follows, we discuss useful OMT structures, micro-transactions, verification/validation of micro-transactions;

and verifiable execution (and output) of micro-transactions that constitute each of the sub-processes $\mathcal{P}_0, \mathcal{P}_1, ...\mathcal{P}_3$ in an SQDM protocol. For ease of explanation, we use the following simplified map to discuss the operations of the sub-processes in an SQDM protocol.

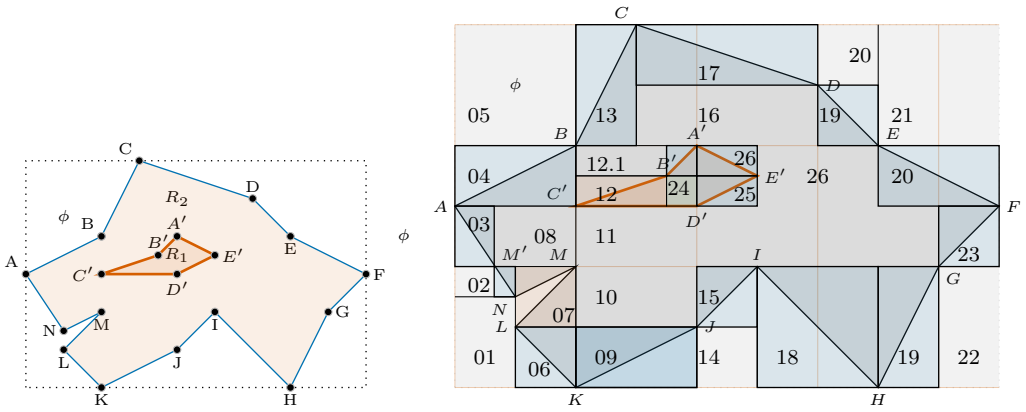

Figure 4.2

Left: A polygon $ABCD..MNA$ on the map whose interior region has an island polygon $A'E'..C'B'A'$. Right: The map on the left is transformed into a mesh of bounding blocks which captures the bounding segments of the polygons.

Furthermore, we will use notations and symbols as follow: a '=' or ':=' is an assignment of a variable to a value; '$\rightarrow$' is update of a value on the left to the value on the right; '$\leftarrow$' is setting an existing variable by a new value; a symbol preceding ':' is used to annotate object or value on the right and with other variables as usual semantics.

4.2.1 Pre-Processing $\mathcal{P}_0$

An SQDM protocol bootstrap by accepting a counter-clockwise (CCW), sequence of boundary points of a simple polygon in a 2D plane. In other words, a polygon with n sides is described by $n+1$ points, where the last point (with the highest point index) and the first point (with the smallest point index) have the same geographic coordinates (x, y). As shown in Figure 4.2 (Left), R_2 is bounded by a sequence of vertices $A, B, C, ..N, A$, and the island region R_1 is inside R_2, and is bounded by points $A', E'..C', B', A'$. All the space outside the regions R_2 and R_1 is a Universe ϕ. Two consecutive distinct points $p(x_1, y_1)$ and $p'(x_2, y_2)$ is a segment (or only line segment) that separates two regions, ρ_a, and ρ_b in a 2D plane. ρ_a is the region above the segment pp', and ρ_b is the region code for the region below the segment pp'. For instance, segment CD separates region $\rho_a := R_1$ just above it, and region $\rho_b := R_0$ below it; segment QR separates region $\rho_a := R_0$ above it, and region $\rho_b := R_1$ just below it, and so on.

In general, an SQDM process starts by taking input boundary points $L_{R_0 \cup .. R_n} = \{\mathcal{L}_0 : \{p_i : p_i \in R_0\} ... \mathcal{L}_n : \{p_i : p_i \in R_n\}\}$. For an input in Figure 4.2 (Left), all together, boundary points in two polygons R_1 and R_2 are collected in two collections $\mathcal{L}_1$ and L_2 given as:

$$L_{R_1 \cup R_2} = \{L_1 : \{A, B, C..A\}, L_2 : \{A', B', ...E'\}\}$$

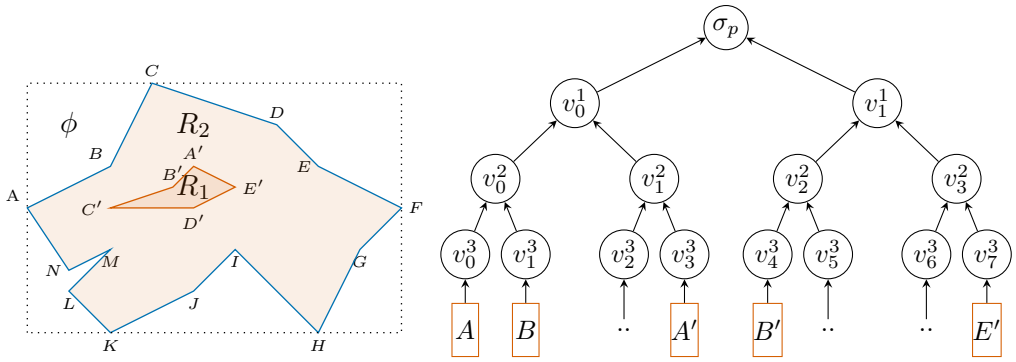

Figure 4.3

(Left): An input polygon is a sequence of points. (Right): All the points in the input polygons are added to a 1D-OMT $\mathcal{T}_p$ that represents a dynamic polygonal object.

A point p in a polygon m is identified by what is called SQDM point defined as:

$$p = m \parallel n \parallel (x, y)$$

It conveys three distinct properties of a point in the polygon.

1. a polygon number m, which is even for enclosing polygons and odd for excluding polygons. For example, for the region R_1, m is even, while for the region R_3, this value is odd.
2. the point index n, which increases monotonically on CCW traversal of boundary points in any polygon m, and
3. the geographic coordinate (x, y) of the point.

This process $\mathcal{P}_0$ invokes micro-transaction $\mathsf{AddPoint}(\mathsf{p}, \mathsf{q}, \mathsf{q}')$ to add all points p in a polygon R_i to the boundary-point OMT with root σ_p. The first point is added by invoking $\mathsf{AddPoint}(\mathsf{p}_0, \mathsf{q}, \mathsf{q}')$, where $q = q' = 0$, with dynamic root $\sigma_p = h(q, q')$. Subsequent points are inserted until last the point p_{n+1} is encountered. A complete 1D

OMT $\mathcal{T}_p$ represents the complete region bounded by points in a CCW. A completion of boundary-point OMT triggers process $\mathcal{P}_1$ for the construction of a mess of BB. For the example input polygons R_1 and R_2, the leaves of $\mathcal{T}_p$ can be listed as:

$$\begin{array}{l}\{(2 \parallel 0 \parallel (x_0, y_0), 2 \parallel 1 \parallel (x_1, y_1), x_0 \parallel y_0), \\ (2 \parallel 1 \parallel (x_1, y_{11}), 2 \parallel 2 \parallel (x_2, y_2), x_1 \parallel y_1), \\ \qquad \vdots \\ (2 \parallel 15 \parallel (x_{15}, y_{15}), x_0 \parallel y_0, 2 \parallel 0 \parallel (x_0, y_0))\}\end{array}$$

From now on, we use a notation such as pp' to represent a segment connecting points p and p'; $pp' \cdot \rho_a$, and $pp' \cdot \rho_b$ to retrieve region labels associated with any segment connecting two points p and p'.

4.2.1.1 SQDM Map Construction

After completely adding boundary points into a boundary-point OMT $\mathcal{T}_p$, the root of the tree is σ_p. The completion initiates an SQDM map construction process. In this step, an input boundary-point OMT $\mathcal{T}_p$ for a region R is used to incrementally build a 2D template map representing the given region. The 2D map is a set of bounding boxes that can contain the partial or complete segments in the input polygon represented by leaves of 1D-OMT. It is represented by a 2D-OMT $\mathcal{T}_B$, whose dynamic root is σ_B. Each entry/record in the map OMT is a bounding box (BB) is represented by a 5-tuple

$$(x_1, y_1, x_2, y_2, v)$$

The initial state of the blockchain for an SQDM protocol before the start of a map construction process is recorded at time t as

$$(\sigma_p, \sigma_B \rightarrow 0)_t$$

Map construction incrementally modifies the existing leaf nodes in both OMTs $\mathcal{T}_p$ and $\mathcal{T}_b$ to progress the roots σ_p and σ_B to progress the current state- $S_0(\sigma_p, \sigma_B) \rightarrow S_1(\sigma'_p, \sigma'_B)$.

The process for map construction follows the following steps:

- Bootstrapping by initializing a blockchain ledger with an empty bounding box $[x_l, y_l, x_h, y_h]$that covers the complete globe. It is the first entry to a 2D-OMT $\mathcal{T}_B$ with dynamic root σ_B, and the value of the dynamic root is $\sigma_B \rightarrow h(x_l \parallel y_l \parallel x_h \parallel y_h \parallel 0)$.
- Clip (split or slice, break) existing BB to initiate the insertion of constituent BBs. A BB can be divided into two parts by introducing either a vertical line or a horizontal line through the bounding box. For example, a bounding box $(1, 5, 9, 8)$ split by vertical line $x = 3$ gives two bounding boxes: i) $(1, 5, 3, 8)$ and ii) $(3, 5, , 9, 8)$. Split by a horizontal line $y = 6$ gives two different boxes: i) $(1, 5, 9, 6)$ and ii) $(1, 6, 9, 8)$. A micro-transaction $\mathsf{SplitBB}(\mathsf{x_l}, \mathsf{y_l}, \mathsf{x_h}, \mathsf{y_h}, \mathsf{x'}, \mathsf{y'})$ is executed to split an input BB (x_l, y_l, x_h, y_h) either vertically along $x = x'$ or horizontally along $y = y'$. In addition, if the split axis intersects any segment in boundary-point OMT $\mathcal{T}_p$, invoke $\mathsf{Interpolate}(\mathsf{p}, \mathsf{p'}, \{\mathsf{x'}, \mathsf{y'}\})$ to insert intersection points of the split axis and all the line segment joining points $p \in \mathcal{T}_p$ and $p' \in \mathcal{T}_p$.

The BBs constructed are inserted into 2D-OMT $\mathcal{T}_B$. The choice where to split depends on the strategy that minimizes the number of BBs that can completely bound/contain all the segments in an input boundary-point OMT $\mathcal{T}_p$. Specifically, the map construction follows two major steps:

1. Invoke μ-transaction $\mathsf{InitGlobe}(\mathsf{x_l}, \mathsf{y_l}, \mathsf{x_h}, \mathsf{y_h}, \phi, \sigma_\mathsf{p}, \sigma_\mathsf{pd})$ to initialize region OMT $\mathcal{T}_B$. The rectangular region $(x_l, y_l) \times (x_h, y_h)$ covers the complete region for an

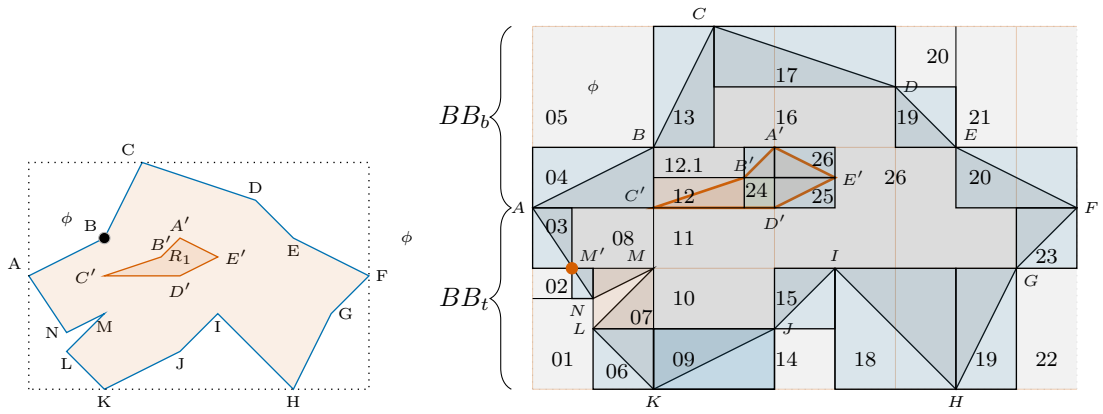

Figure 4.4

An input polygon $ABC...MA$ is divided into a mesh of bounding blocks called BB. A strategic selection of split axis (horizontal or vertical) produces the minimal number of blocks that sufficiently represent the region enclosed by boundary points.

input polygon(s). It also sets the dynamic root σ_{pd} of $\mathcal{T}_{pd}$ to terminal root value σ_p, which means initiation of construction of map $\mathcal{T}_B$ for the region $\mathcal{T}_p$.

2. Invoke μ-transaction $\mathsf{SplitBB}(\mathsf{x_l}, \mathsf{y_l}, \mathsf{x_h}, \mathsf{y_h}, \mathsf{x'}, \mathsf{y'})$ to split existing BB along appropriate axis ($x = x'$ or $y = y'$) into two children BBs.

3. Replace the original BB by two children BBs, and update the root of 2D-OMT $\mathcal{T}_B$.

4. Invoke μ-transaction $\mathsf{Interpolate}(\mathsf{p}, \mathsf{p'}, \{\mathsf{x'}, \mathsf{y'}\})$ to insert a new point $(x_i, y_i) = I(p, p', [x_i, y_i])$, which is an intersection point between segment pp' and split axis along $x = x'$ or $y = y'$.

5. Repeat (1), (2), and (3) until a BB contains at most 2 input segments (or their fragments).

6. Update current blockchain state $(\sigma_p, \sigma_B)_t$ to next state $(\sigma'_p, \sigma'_B)_{t'}$.

With respect to the same input polygons R_1 and R_2, as shown in Figure 4.4, we outline a map construction method in this section. The map construction process starts by initializing the region OMT by inserting a bounding box $WW'ZZ'$ as the first leaf in the tree. In other words, we execute a μ-transaction $\mathsf{InitGlobe}(\mathsf{W_x}, \mathsf{W_y}, \mathsf{W'_x}, \mathsf{Z_y}, \phi)$

to insert a BB that covers an input polygon(s) R_1 and R_2. It is followed by vertically splitting the initial BB along $x = K_x$ to replace it by two BBs, $BB_0 : WKK'Z$ and $BB_1 : KXYK'$. This particular split causes to partition the input segments into two groups:

$$\begin{array}{ll} BB_0: & \{AB, AN, NM, LK\} \\ BB_1: & \{BC, CD, DE, EF, CF, HC, IH, JI, KJ\} \end{array}$$

The split axis does not intersect through any input segment. Hence μ-transaction Interpolate(.) is not executed. Similarly, block BB_0 is split horizontally along axis $y = A_y$ to produce top block BB_t that contain segments $\{AB\}$ and bottom block BB_b that contains segments $\{AN, NM, LK\}$. This split also avoids $\mu-$transaction Interpolate(.). To proceed further, the block BB_b can be split horizontally along $y = M_y$ that ultimately splits the segment AN into AM' and $M'N$ at A' (red filled circle). This is the time when micro-transaction Interpolate(A, N, y = $\mathsf{M_y}$) is invoked to introduce a new point M' in boundary-point OMT $\mathcal{T}_p$. The effect is that: each of the μ-transaction replaces a BB leaf in the region OMT by two new BBs, and potentially replace a leaf in boundary-point OMT by two new points. The operation InitGlobe and SplitBB μ-transaction update the root σ_{BB} of region OMT $\mathcal{T}_{BB}$, while the operation Interpolate updates the root σ_p of the boundary-point OMT $\mathcal{T}_p$. A similar procedure is applied to the entire block on the right of the initial axis $x = K_x$. This produces the complete 2D mesh, as shown in Figure 4.4(b).

4.2.1.2 Types of BBs

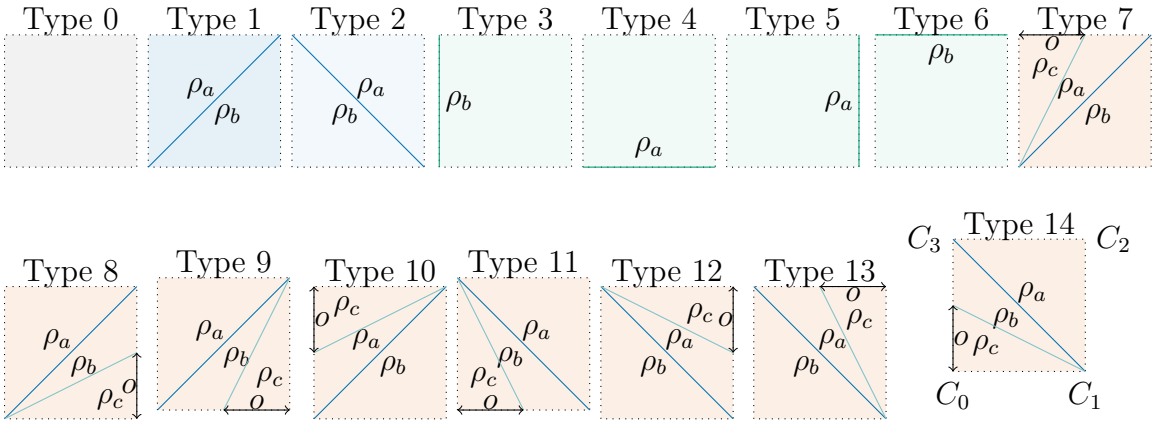

Figure 4.5

Types of BBs

The map construction process discussed in the previous section produces different types of BBs depending upon the orientation of the segments that a BB contains. As shown in Figure 4.4, the gray shaded blocks such as **01**, **05**, **11**, **10**, **16**, **20**, **21**, **22**, and **26** contain no line segment of the input polygons. They are categorized as Group-0 BBs. The blue and green shaded blocks **03**, **04**, **06**, **09**, .., **15**, **25**, **26**, **17**, **24** contain a single segment as the block's main diagonal (positive or negative) are categorized as Group-I blocks. In case two consecutive segments such as NM and LM meet at an acute or right angle, a block such as **07** and **12** (red shaded) is constructed. Such blocks are categorized as Group-II blocks. It is important to note that these are the ideal blocks that a map construction process must produce. Specifically, as shown in Figure 4.5 the Group-I blocks can be further identified as six different types:

- Type-1, a segment in a polygon becomes a positive diagonal of a containing BB.
- Type-2, a segment in a polygon becomes a negative diagonal of a containing BB.
- Type-3, a segment in a polygon becomes vertical left side of a BB.
- Type-4, a segment in a polygon becomes vertical bottom side of a BB.
- Type-5, a segment in a polygon becomes vertical right side of a BB.
- Type-6, a segment in a polygon becomes vertical top side of a BB.

As shown in the same Figure 4.5, the Group-II blocks are differentiated into 8 types as:

- Type-7, contains two segments meeting at the lower-left corner C_1 of the containing BB, where a longer becomes a positive diagonal and shorter becomes support segment with horizontal offset of o from the corner.
- Type-8, contains two segments meeting at the lower-left corner C_1 of the containing BB, where a longer becomes a positive diagonal and shorter becomes support segment with vertical offset of o from the corner.
- Type-9, contains two segments meeting at the upper right corner C_2 of the containing BB, where a longer becomes a positive diagonal and shorter becomes support segment with horizontal offset of $o$from the corner.
- Type-10, contains two segments meeting at the lower-left corner C_2 of the containing BB, where a longer becomes a positive diagonal and shorter becomes support segment with vertical offset of o from the corner.
- Type-11, contains two segments meeting at the upper left corner C_4 of the containing BB, where a longer becomes a negative diagonal and shorter becomes support segment with horizontal offset of o from the corner.
- Type-12, contains two segments meeting at the upper left corner C_4 of the containing BB, where a longer becomes a negative diagonal and shorter becomes support segment with vertical offset of o from the corner.
- Type-13, contains two segments meeting at the lower right corner C_3 of the containing BB, where a longer becomes a negative diagonal and shorter becomes support segment with horizontal offset of o from the corner.

- Type-14, contains two segments meeting at the lower right corner C_3 of the containing BB, where a longer becomes a negative diagonal and shorter becomes support segment with vertical offset of o from the corner. The regions that the diagonal separates is designated as ρ_a ρ_b, and the region that the shorter segment separates are either of ρ_a and ρ_b and ρ_c.

Together a collection of BBs is a template SQDM R_{SQDM} for an input map.

$$\mathbb{R}_{SQDM}: \quad \{01, 02, 03, 04, .., 20, 21, 22, 23, 24\}$$

Note that each BB in $\mathbb{R}_{SQDM}$ is a leaf node in 2D- OMT T_B with dynamic root σ_B. Specifically, a BB in T_B is of the form:

$$(x_l, y_l, x_h, y_h, v)$$

4.2.2 Segment to BB Mapping $\mathcal{P}_1$

Every segment pp' in $\mathcal{T}_p$ produced by $\mathcal{P}_0$ can now be mapped to a one of Type 0, Type 1, ..,Type 14 **BB**s in a template SQDM, $\mathbb{R}_{SQDM}$ generated by process $\mathcal{P}_1$.

A sequence of μ-transaction $\mathsf{MapLine}()$ is invoked to map all the segments in 1-D OMT $\mathcal{T}_p$ to one of the BB in T_B. A μ-transaction $\mathsf{MapLine}(\mathsf{p}, \mathsf{p}', \mathsf{x_l}, \mathsf{y_l}, \mathsf{x_h}, \mathsf{y_h}, \mathsf{v})$ updates

1. the value v $BB : [x_l, y_l, x_l, y_l, v] \in \mathbb{R}_{SQDM}$ by the encoding of the mapped segment $pp' \in \mathcal{T}_p$.
2. the dynamic value $\mathsf{START} = \mathsf{0}$ and $\mathsf{END} = \phi$ to the endpoint of the segment pp' to track the completion of the mapping process. The mapping must terminate whenever $\mathsf{START} = \mathsf{END}$.

In fact, an encoding for a segment captures sufficiently minimum information that represents the segment. An encoding v for a segment is formulated according to the

type of block $c = \{0, 1, 2...14\}$ being mapped to, vertical offset o of small segment s (if any), and region codes ρ_a,ρ_b, ρ_c associated with the segment pp'.

$$
\begin{aligned}
&\text{Type 0} \quad v = (c \parallel \rho_a \parallel \rho_b \parallel \rho_c \parallel o) = 0 \parallel 0 \parallel 0 \parallel 0 \parallel 0 \\
&\text{Type 1} \quad v = (c \parallel \rho_a \parallel \rho_b \parallel \rho_c \parallel o) = 1 \parallel \rho_a \parallel \rho_b \parallel 0 \parallel 0 \\
&\text{Type 2} \quad v = (c \parallel \rho_a \parallel \rho_b \parallel \rho_c \parallel o) = 2 \parallel \rho_a \parallel \rho_b \parallel 0 \parallel 0 \\
&\text{Type 3} \quad v = (c \parallel \rho_a \parallel \rho_b \parallel \rho_c \parallel o) = 3 \parallel \rho_a \parallel \rho_b \parallel 0 \parallel 0 \\
&\text{Type 4} \quad v = (c \parallel \rho_a \parallel \rho_b \parallel \rho_c \parallel o) = 4 \parallel \rho_a \parallel \rho_b \parallel 0 \parallel 0 \\
&\vdots \\
&\text{Type 7} \quad v = (c \parallel \rho_a \parallel \rho_b \parallel \rho_c \parallel o) = 7 \parallel \rho_a \parallel \rho_b \parallel \rho_c \parallel o \\
&\text{Type 8} \quad v = (c \parallel \rho_a \parallel \rho_b \parallel \rho_c \parallel o) = 8 \parallel \rho_a \parallel \rho_b \parallel \rho_c \parallel o \\
&\vdots
\end{aligned}
\tag{4.1}
$$

Each mapping updates the root σ_B of the region OMT. After the completion of this process, the values of the roots of the two OMTs $\mathcal{T}_B$ and $\mathcal{T}_p$ are (σ_{pd}, σ_B) on which a BN reaches a consensus on the integrity of the input polygon points as well as corresponding 2D map constructed. In short, the process of adding boundary points into a 1D-OMT and construction of the template map and the mapping of the segments to the map can be seen as series of progression of the process states captured by the roots of the two OMTs.

$$(\sigma_p \leftarrow 0, \sigma_B \leftarrow 0) \xrightarrow{\mathsf{AddPoint(p_0)}} (\sigma''_p, 0)_1 \xrightarrow{\mathsf{AddPoint(p_1)}} (\sigma'_p, 0)_2$$

$$\ldots \xrightarrow{\mathsf{(AddPoint(p_n)}} (\sigma^*_p, 0)_{n+1}$$

$$(\sigma_{pd} \leftarrow \sigma^*_p, \sigma_B \leftarrow 0) \xrightarrow{\mathsf{InitGlobe}} (\sigma_{pd}, \sigma'_B)_1 \xrightarrow[\mathsf{Interpolate(pp',q)}]{\mathsf{SplitBB}(B_0,\mathsf{[x',y']})} (\sigma'_{pd}, \sigma''_B)_2$$

$$\ldots \xrightarrow[\mathsf{Interpolate(pp',q)}]{\mathsf{SplitBB}(B_n,\mathsf{[x,y]})} (\sigma^*_{pd}, \sigma^*_B)_{n+1}$$

$$(\sigma^*_{pd}, \sigma_{Bm} \leftarrow \sigma^*_B) \xrightarrow{\mathsf{MapLine(p_0,p_1,B)}} (\sigma^*_{pd}, \sigma'_{Bm})_1 \xrightarrow{\mathsf{MapLine(p_1,p_2,B')}} (\sigma^*_{pd}, \sigma''_{Bm}))_2$$

$$\ldots \xrightarrow{\mathsf{MapLine(p_n,p_0,B''')}} (\sigma^*_{pd}, \sigma^*_{Bm}))_{n+1}$$

4.2.3 Resolving Queries $\mathcal{P}_2$

Once the input segments are mapped to one of the BBs in an SQDM, the system is in the state of responding queries from any client. The process for answering (resolving) queries is as follows. Given a query point $q(x, y)$, the BB $[x_1, y_1, x_2, y_2, v]$ that contains the point must satisfy $x_1 \leq x < x_2, y_1 \leq y < y_2$ is identified in a SQDM map $\mathcal{T}_B$.

If the response block is of a Group I, the region codes ρ_a, and ρ_b associated with the block value v is sufficient to determine the location of the query point. It is simply done by checking if the query point is above or below a diagonal of the block.

Given a response block in Group II $\{7, 8, 9, 10..14\}$, to find the region in which the point (x, y) falls; it is sufficient to decide on 'if (x, y) is BELOW the diagonal of the block.' Suppose that a Group II block $[x_1, y_1, x_2, y_2, v = [c \parallel \rho_a \parallel \rho_b \parallel \rho_c \parallel o]]$ is a response block, the two lines corresponding to different Type II blocks $c = \{7, 8, 9, 10, ...14\}$ can be resolved as:

Main Diagonal	Support Segment	BB Type
(x_1, y_1)—(x_2, y_2)	(x_1, y_1)—$(x_1 + o, y_2)$	7
(x_1, y_1)—(x_2, y_2)	(x_1, y_1)—$(x_2, y_1 + o)$	8
(x_1, y_1)—(x_2, y_2)	$(x_2 - 0, y_1)$—(x_2, y_2)	9
(x_1, y_1)—(x_2, y_2)	$(x_1, y_2 - o)$—(x_2, y_2)	10
(x_1, y_2)—(x_2, y_1)	(x_1, y_2)—$(x_1 + o, y_1)$	11
(x_1, y_2)—(x_2, y_1)	(x_1, y_2)—$(x_2, y_2 - o)$	12
(x_1, y_2)—(x_2, y_1)	$(x_2 - o, y_2)$—(x_2, y_1)	13
(x_1, y_2)—(x_2, y_1)	$(x_1, y_1 + o)$—(x_2, y_1)	14

(4.2)

An ability to check if a query point (x, y) is below both lines or above both lines or in-between the two lines is sufficient to determine the region that contains the query point.

4.2.4 Map Construction By Slab Decomposition Method

Similar to a slab decomposition method described in [5], steps for map construction is described as the following steps:

i) **Identify Ux**: Identify all unique X-coordinates in all endpoints of segments in L to find vertical columns (slabs) for the map. As shown in figure 4.2, we have 10 unique x-values $U_x\{A_x, B_x, C_x, ..H_x, A'_x, ..F_x\}$. The vertical lines passing through these points define vertical slabs $S_0, S_1...S_9$ over the input map. Each vertical slab can be identified by a pair of vertical edges that pass vertically through adjacent X-coordinates in U_x. Vertical slabs in the input map be annotated as:

$$S_1[A_x, N_x], S_2 : [N_x, K_x]..., S_9[G_x, F_x]$$

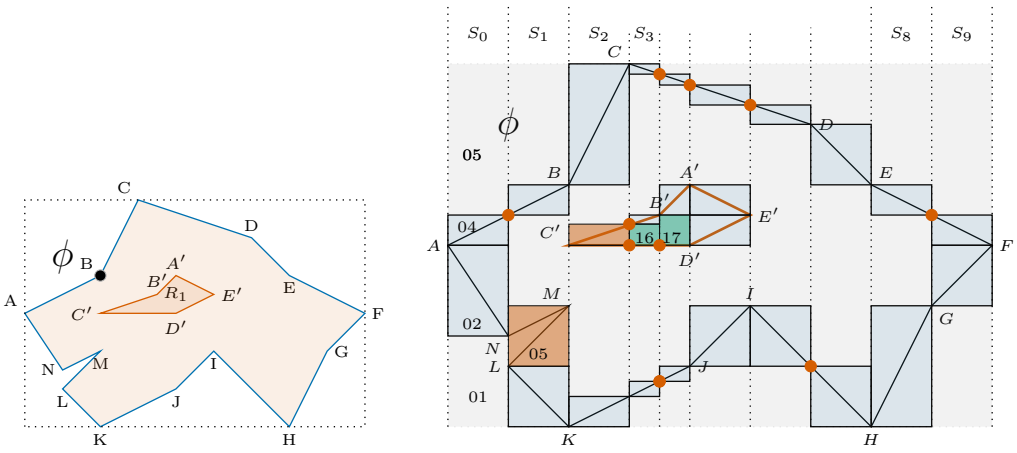

Figure 4.6

An input polygon $ANM...BA$ is divided into a mesh of bounding blocks (BBs) using the slab decomposition technique. Three types of blocks are represented by gray shaded, blue shaded, and red shaded rectangles.

ii) **Identify Bounding Boxes (BBs)**: Split all input segments in $pp \in \mathcal{T}_p$ at points where they intersect the vertical axis through unique X-values in U_x. For in-

stance, such split points are the red filled circles in Figure 4.6. Segment AB intersects vertical edge through N_x, hence it is split into segments AN' and $N'B$. Similarly, segment KJ is split at C_x, B'_x, D'_x to generate three splits, and so on. All splits produced from original segments in $L_{R_1 \cup R_2}$ collected for respective vertical slabs, $S1..S9$ are:

$$\begin{array}{ll} S_0 : A_x\text{—}N_x & \{AN, AN'\} \\ S_1 : N_x\text{—}K_x & \{N'B, NM, LM, LK\} \\ \quad \vdots & \\ S_9 : G_x\text{—}F_x & \{FG', GF\} \end{array}$$

iii) **Identify Non-Overlapping Bounding Box (BB)**:

Identify a non-overlapping bounding box (**BB**) enclosing each of the split segments in $\mathcal{T}_p$. As overlaid in Figure 4.6, BB around split AN is 02; around CD' is $bBB2$, BB around LM and NM is 05, and so on. In the case of 05 BB, two segments in a slab meet at a single vertex (such as E) at an acute angle, their blocks overlap each other in both X and Y-projection. From Figure 4.6, along the boundary of the input map, we identify types of BBs, as explained in the previous section.

4.2.5 Constrained Mapping

The constraints imposed on the mapping ensure that the polygon describing the region is simple and complete. Specifically, the following 2 constraints, viz.,

1. a single line mapped to a BB should be a side or a diagonal.
2. 2 adjacent boundary lines may be mapped to a BB, if one of them is a diagonal.

ensure that boundary lines can not cross each other inside BBs.

The third constraint imposed is the uniqueness of interpolated points. A boundary line is specified by 2 points - say (p_j, p_{j+1}). Given 2 adjacent boundary lines (p_{i-1}, p_i) and (p_i, p_{i+1}), the point p_i is an interpolated point is p_i lies on the line connecting p_{i-1} and p_{i+1}. Ensuring the uniqueness of interpolated points ensures that boundary lines cannot cross each other at BB corners.

The fourth constraint is that mapping is to be performed sequentially, in the CCW order of boundary lines. The mapping is complete when the start and end points are the same. Together, all constraints ensure that a list of boundary lines represents a valid polygonal region.

At the end of successful mapping, a list of boundary lines representing a region is converted to a list of BBs. Each item in the BB list i) specifies a bounding X and Y coordinates, and ii) explicitly identifies 0, 1 or 2 lines mapped to the BB. The main advantage of converting a list of lines to a list of BBs is that any query regarding any point (x, y) inside the original dashed BB can be answered by examining a single BB. For example, in blue BBs 04 and 17, any point that falls below the diagonal is inside the region; in blue BBs 09, 19 and 23, all points above the diagonal are inside the region. In BBs 24, all points above the boundary line $(C'D')$ (the bottom side) are inside the island region. In the red BB 07 in Figure 4.7 (Right), only points that lie in between the two mapped lines are outside the regions bounded by the boundaries.

To see the need for the third constraint – uniqueness of interpolated points – consider the non-simple polygon ABCD in Figure 4.7(Left) which obviously does

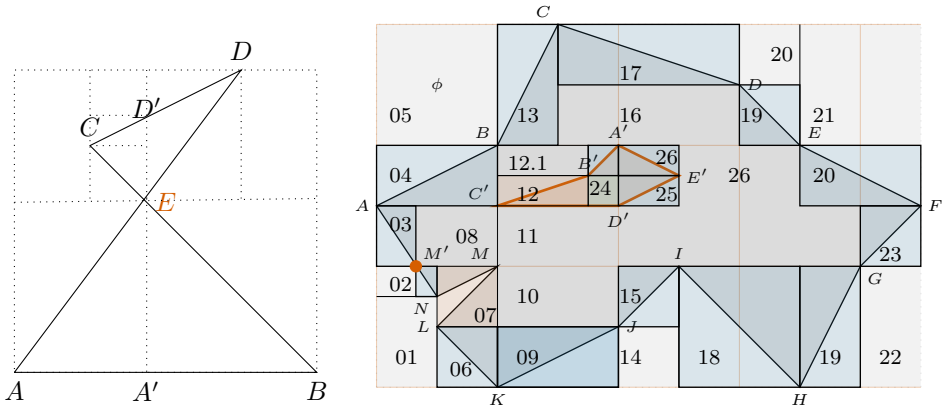

Figure 4.7

Left: Effect of Ordering Boundary Points on Validity of a Region. **Right**: Two input polygons $ABC...MA$ and $A'E'..C'E'A'$ are divided into a mesh of bounding blocks called BBs. The mesh is in deed an SQDM for the two polygons.

not describe a valid region. By splitting boundary lines AB at A', BC at E, CD at D', and DA at E, all boundary lines of this obviously invalid region can still be mapped to BBs (as the boundary does not intersect *inside* any BB). Specifically, the boundary lines can now be mapped in the order AA', $A'B$, BE, EC, CD', $D'D$, DE and EA. However, the points added by interpolation are A', E, D' and E, which are not unique (as E occurs twice). Thus, the third constraint will cause the mapping to fail.

It is important to note that the situation would be different if the original region was described instead as ABEDCEA – by a sequence of boundary lines AB, BE, ED, DC, CE and EA. In this case only 2 points need to be added by interpolation (A' and D'), to perform the mapping in the order AA', $A'B$, BE, ED, DD', $D'C$, CE, and EA. Mapping this sequence of boundary lines will be successful – as it should be, as the

boundary lines do describe a valid region (two triangular regions meeting at corner E). Note that while we have 2 instances of point E, it is *not* a point added by interpolation. In summary, the three constraints ensure that a given vector of points represents a valid region.

4.2.6 Necessity and Sufficiency

One of the primary motivations of splitting complex processes into multiple transactions is to reduce the complexity of individual transactions. It is obviously advantageous to limit the number of lines in each BB to reduce complexity – after all, we pay less care about the number of transactions and more about the simplicity of transactions. Unfortunately, a minimum of 2 lines is *necessary* in scenarios where 2 adjacent boundary lines have overlapping projections in *both* X and Y directions. Fortunately, limiting BBs to 2 lines ("red" BBs) is also sufficient.

To see why this is indeed the case, consider the scenario in Figure 4.8(a), where several lines meet at a corner. It might appear at first glance that how many every subdivisions are performed to split lines or BBs, a BB at the lower-left corner at which all lines meet will still have several lines mapped to it.

However, only a finite number of bits can be used to represent x and y coordinates. For example, describing coordinates using unsigned 32-bits is sufficient to realize a worst-case resolution error (at the equator) of less than 1 cm. With finite precision, in the smallest possible BB $(x, y, x+1, y+1, v)$ at the lower-left corner, all lines will

have to map to the bottom side or the diagonal, or the left side of the BB (as in Figure 4.8(b)).

In other words, at most 3 lines may need to be mapped to a BB. However, we have the option to map the bottom of the BB as the top side of the BB below or the left side as the right side of the BB to the left. In the worst-case scenario, only 8 lines will need to be mapped to 4 adjacent BBs (or 2 each), as shown in Figure 4.8(c).

A consequence of limiting BBs to 2 lines is that we can not add any more lines to a BB that is already red. By splitting a red BB into 3 (see Figures 4.8(d) and (e)), it can be converted into a smaller red BB, a blue BB, and a clear BB. This operation, which renders the smaller line of the larger red BB as the diagonal of the smaller red BB (transaction SplitBB2()) can be repeated any number of times (if necessary) to reduce the size of the remaining red BB (until new boundary lines to be added fall entirely in clear or blue BBs). In the next section, we discuss a method to reduce Group-II BBs into smaller Group-I BBs further.

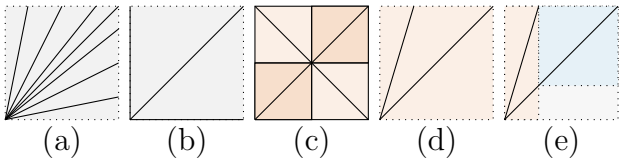

Figure 4.8

(a)-(c) Necessity and Sufficiency of "red" BBs with 2 line segments. (d)-(e) a red BB split into 3 BBs.

4.2.6.1 Reducing Group-II blocks

Group-II $\{7, 8, 9, 10, ..13, 14\}$ blocks encloses two or more split segments. It is desirable that this type of block is split multiple times to create a few Type 0, Group-I blocks. This breaking down shall facilitate assigning block value v with a minimal piece of information (potentially segment/region information). As shown in the adjoining Figure 4.9, a Type-12 block with two segments AB and AC is broken down into two Type $\{1, 2\}$ blocks (blue shaded blocks 02 and 03), and two Type 0 (gray blocks 01 and 04) blocks.

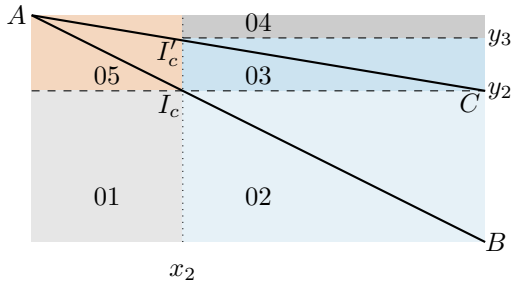

Figure 4.9

A Type 12 block in an SQDM is reduced to few Type 0, Type 1, and smaller Type 12 blocks.

The first split is done to segment AB at point $I_c(x_2, y_2)$intersected it by horizontal line through right endpoint C of shorter segment AC. This split created a Type 0 block 01, and a Type 1 block 02 around split I_cB. Having known x_2, shorter segment AC is again split at $I'_c(x_2, y_3)$. This split creates a Type II block (red shaded $rBB1$)

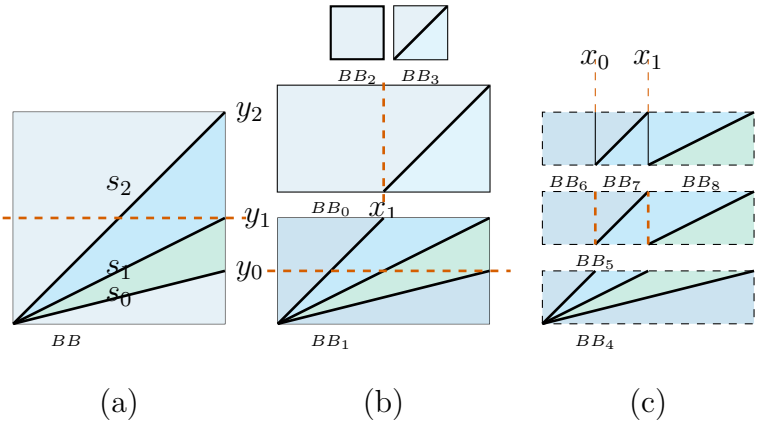

Figure 4.10

A Type-8 BB is clipped horizontally through y_1. It produces $\{BB_0, BB_1\}$. Block BB_0, when clipped vertically through x_1, produces smaller BBs $\{BB_3, BB_4\}$. Block BB_1 contains 3 splits, it is clipped horizontally through y_0 produces $\{BB5, BB_4\}$. BB_5 contains two splits, which will be split vertically through x_1 and x_0 produces $\{BB_6, BB_7, BB_8\}$. Block BB_4 is a rotated version of initial block BB, which can be clipped first vertically and then horizontally recursively.

containing two splits AI'_c and AI_c, and a Type I block 03 containing split I'_cC; and a small Type 0 block $wBB2$.

Similarly, as shown in Figure 4.10, a Type-8 BB contains three segments s_0, s_1, s_2. It is further split into multiple Type 0, Group-I, and a smaller Group-II block. Thus a rule of thumb for clipping BBs is that reduced blocks must be of any of Type $\{0, 1, 2, .., 14\}$.

4.3 Related Work, Result and Conclusion

SQDM processes were applied to the real-world geo-spatial data in ESRI Shapefile [28] format to an SQDM. Shapefiles representing US states, US Congressional Districts, and MS counties were converted to an SQDM representation. A national,

state or county boundary or a parcel of geographic space is represented by a set of geographic coordinates $P(\lambda, \phi)$, where λ represents the value of geographic latitude defined such that $-90^\circ \leq \lambda \leq 90^\circ$; and ϕ represents the value of geographic longitude defined such that $180^\circ \leq \phi \leq +180^\circ$. The system of assigning these latitude and longitude values is based on a geographic coordinate system (GCS), which is based on a standard geodetic systems such as North American Datum (NAD) 1983, WGS 84. The geographic coordinates can be transformed to represent location on a Euclidean plane (two-dimensional plane) using a projected coordinate system (PCS) such as transverse Mercator or Pseudo-Mercator or Universal Transverse Mercator (UTM). Most often, geographic regions such as countries, States, or cities are referenced in different geographic/planar coordinate systems (GCS). These maps data are transformed to a common reference coordinate system [64], to ensure that data represents a common co-ordinate reference system (GCS or PCS).

The boundary map of the USA consists of 56282 segments. Three methods for constructing SQDM blocks for the boundaries of the USA and its states were investigated to find out the size of the μ-transactions for SQDM protocol. The number of unique x-values is 55000. During the construction of an SQDM for the continental map of the USA, it is necessary to split the boundary segments at unique X-ordinates. The total number of such split is 627084. Figure 4.11 shows the split count frequency of segments that suffer splitting. The diagram indicates that the majority of segments have split less than 100. The maximum number of splits for a segment is 701.

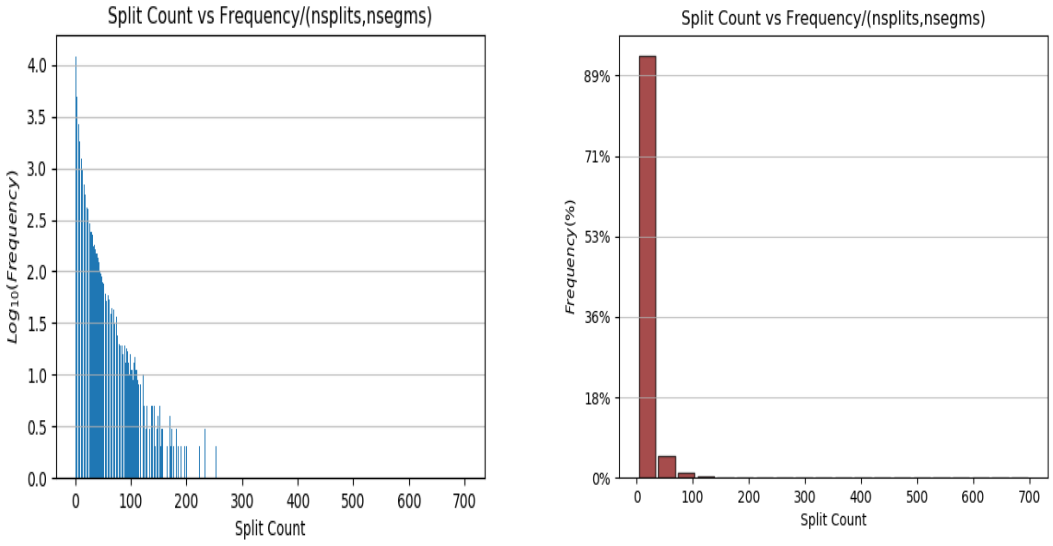

Figure 4.11

The majority of segments have less than 100 splits (left). More than 95 percent of original segments are split into less than 100 points (Middle). (Right) Number of splits and numbers of the bounding boxes for each vertical slab in the US map.

4.3.1 SQDM of the USA States

As an example, the state of Mississippi is a polygon with 2713 points. To perform the restrictive mapping of boundary segments to BBs using map construction method-2,

1. boundary lines needed to be split 32 times;
2. a BB enclosing the entire state was split into 7919 BBs;
3. After mapping boundary lines to BBs (see Figure 7 top-left), 5181 were Type 0 (clear BBs), 2731 were blue/green BBs with a single line (Types $\{1 \cdots 6\}$), 7 were red BBs with two lines (Types $\{7 \cdots 14\}$).

Yazoo county inside Mississippi is a polygon with 713 points. To perform the restrictive mapping of boundary segments to BBs

1. boundary lines needed to be split 35 times;
2. a BB enclosing the entire state was split into 2052 BBs;
3. after mapping boundary lines to BBs (Figure 4.12 (Left: top-right), we had 1317 clear BBs, 722 BBs with a single line, and 13 red BBs.

The following table displays the size of polygon features, the required number of splits, and the numbers of clear, blue/green, and red BBs in the polygon's SQDM.

Table 4.1

The sizes of the input maps of different US states and the number of white and blue BBs in the respective map's SQDMs.

Name	**n(pts)**	n(white BB)	**n(blue BB)**	**Total BBsheight**
MISSISSIPPI	2713	5181	2731	7919
YAZOO (County)	717	1317	722	2052
LOUISIANA	4801	9002	4841	13886
OHIO	2945	5629	2957	8594
OREGON	1961	3692	1959	5656
MONTANA	2327	4355	2325	6682
TENNESSEE	2275	4397	2274	6680
ARIZONA	1416	2663	1415	4080
CALIFORNIA	4382	8336	4383	12740
FLORIDA	5430	10284	5471	15775
IDAHO	2970	5544	2971	8519
KANSAS	1493	2834	1492	4328
KENTUCKY	3102	6008	3103	9117
MASSACHUSE	1822	3505	1873	5399
MARYLAND	6931	13130	6983	20147
MICHIGAN	4744	9022	4784	13816
MASSACHUSE	1822	3505	1873	5399
MINNESOTA	5953	11102	6004	17130
MISSOURI	3031	5800	3029	8832
USA*	56282	106545	56707	163558

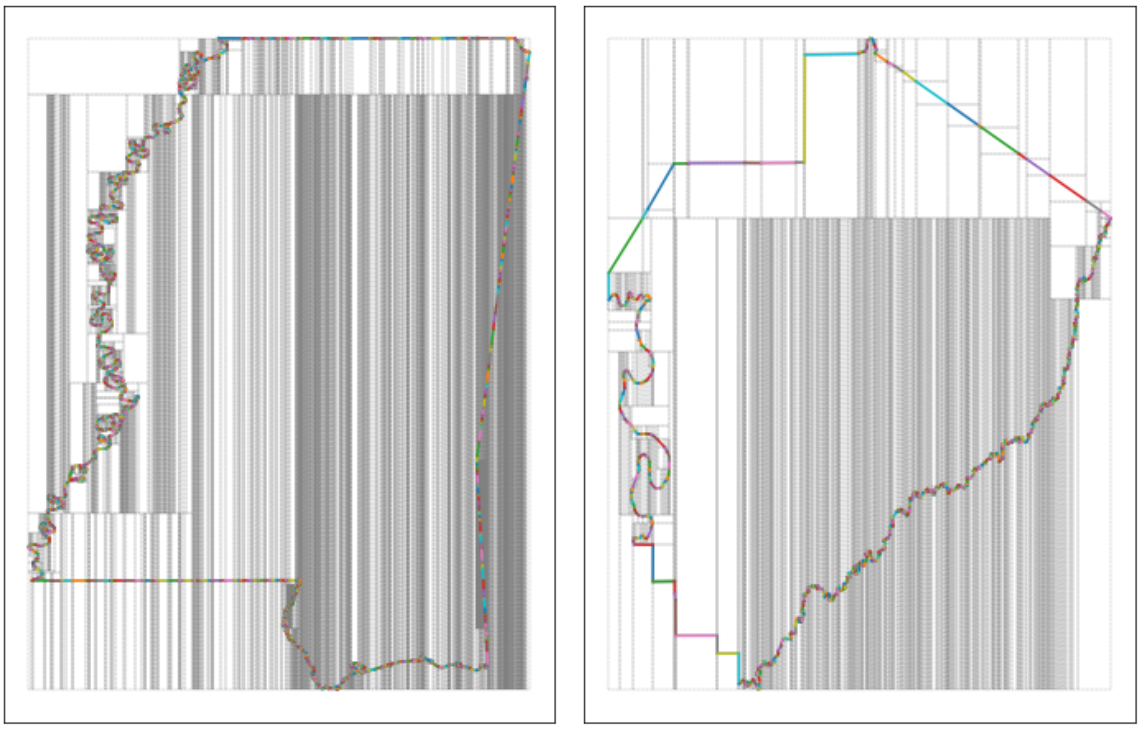

Figure 4.12

Top (left): SQDM for MS state. Top (right): SQDM for Yazoo county.

CHAPTER 5

GEOGRAPHIC REGIONAL DELEGATION PROTOCOL

> With e-currency based on cryptographic proof, without the need to trust a third party middleman, money can be secure and transactions effortless.
>
> *Satoshi Nakamoto*

5.1 Introduction

A Domain Name System (DNS) [52] is a distributed system for a delegation of a namespace to facilitate efficient resolving Internet domain names through separate DNS servers. In a DNS, delegation starts from top-level authority called "root" (/). It delegates namespaces to various zone authorities such as com, org, net, edu. The zone authorities can delegate other distinctive names in their zone to other child-level zones. For example, ownership of a DNS zone `ms.edu` is acquired through delegation from the parent zone `edu`, which was, in turn, due to delegation by the root zone. The owner of zone `ms.edu` can create any number of new names ending with `ms.edu` (for example, `cs.ms.edu`, `ee.ms.edu`, etc.). Such created names can be i) delegated, or ii) bound to different types of information – by creating distinct types of DNS records. In DNS, $x.y.z$ is a child of a zone $y.z$ (or $x.y.z \in y.z$) as

$x.y.z$ **ends with** $y.z$. This hierarchy of DNS zones constructs the DNS namespace. The DNS hierarchy, along with a DNS security protocol (DNSSEC) [36], guarantees authoritative (A) and unbiased response (U) to NS queries.

The DNSSEC associates every zone with a public key. The public key $U_{x.y.z}$ of a zone $x.y.z$ is certified by the parent zone $y.z$, whose public key $U_{y.z}$ is certified by *its* parent zone z, whose public key U_z is certified by the root zone. The public key (say, Φ) of the root zone is assumed to be public knowledge.

Knowledge of the root public key is sufficient to track the authority of a public key U_Z of a zone Z, and consequently, DNS information certified using U_Z (for names that end with Z). The response to a DNS query is a DNS record. Specifically, the steps necessary for authoritatively establishing the integrity of a response R_N for a name N (say, 2 levels of hierarchy below the root), are as follows:

1. verify certificates $\langle R_N \rangle_{U_a}$, $\langle a, U_a \rangle_{U_b}$, $\langle b, U_b \rangle_{U_c}$, and $\langle c, U_c \rangle_{\Phi}$, where Φ is the known root public key; given that the notation $< A, U_A >_U$ is a certificate verifiable using public key U, binding a public key U_A with an entity A.
2. verify that i) name N ends with a; ii) name a ends with b and iii) name b ends with c;
3. if $N \neq a$, confirm that no name ending with N has been delegated by a; for example, if $N = q.w.x.y.z$ and $a = x.y.z$ it is essential to know that $w.x.y.z$ and $q.w.x.y.z$ are *not* DNS zones (delegated by $a = x.y.z$).

The last step, viz., verification of non-existence of delegations, is facilitated by authoritative denial (for providing proof of absence of specific DNS records). In DNS, this is made possible by the NSEC / NSEC3 [21] component of DNSSEC.

Analogous to a zone name in DNS, let's assume that an entity (example, a citizen, Governing body, etc.) has an authority over a geographical region (such as land par-

cel, state, county, city, etc.) For the relation to exist, there must be a domain-specific association between the entity and the geographic region, which enable the entity to claim control (legal right) of over the region. Let us also assume that the entity wishes to transfer the ownership right over to another entity. Under a conventional system, the entity can do so by signing a written contract for a transfer of authority, which should be recognized by a mutually trusted intermediary such as a government office of title registration. The role of a government agency is crucial in terms that it is a mutually trusted third party that authenticates and recognizes the transfer of ownership of the right. It comes with a lengthy and inefficient bureaucratic process. Moreover, delegating a 2D space is not as simple as delegating a scalar value like an Internet domain name. It involves two-dimensional (2D) data representing a region and sub-divisions intended for delegation. It must also preclude the involvement of trusted third parties like title offices. Thus this seemingly trivial process has the following requirements to meet AU requirements.

1. To secure underlying geospatial data of geographic boundaries and associated features (domains) like area, physical structures like restaurants, banks, etc.
2. To prove that an entity has a specific authority over a region to securely delegate partial or complete authority over the region, ensuring that:
    - No overlaps between delegated region;
    - No double-delegation, which means that the same geographic region should not be transferable to multiple entities at the same time, and
    - No unsolicited revocation of a delegated region.
3. To prevent a trusted third party during delegation to bring greater automation for a 2D space delegation.
4. Securely and safely surrender (rollback) delegation once desired.

In the following section (5), we define the protocol called Geographic Regional Delegation Protocol (GRDP) that enables secure, trustworthy context-specific spatial delegation.

5.2 Definition of GRDP

Let P be a non-intersecting, closed geographic region bounded by a series of n boundary line segments. As shown in Figure 5.1, let an entity U_i owns an authority over the surface interior to the region P. Let the relationship be encoded as an association $\langle U_p, P \rangle$. A trustworthy geographic delegation of a sub-division C inside a region P is a transaction T at time t for partial or complete transfer of authority over sub-division C to second entity U_c, thereby transitioning current system state S_t to S_{t+1}. Syntactically, T is defined as:

$$T_t = [P, C, S_t, U_c]_{U_p}, \tag{5.1}$$

where subscript U_i signifies that T is digitally signed by U_p. A successful execution of a delegation transaction is expected to meet the following requirements:

- Avoid double delegation of the child division C by the same authority. It means that once an authority delegates the child division (or any other child division that overlaps with this child division) to another entity, he/she is not able to delegate the same to the second entity.
- Establish ownership of entity U_c authoritatively over a 2D region P without a Trusted Third Party (TTP).
- Authoritative surrendering of a delegation, which guarantees that complete roll-back of a delegation transaction.

In other words, C is a *region* (described by 1 or more polygons) in GRDP. The region C is a "child" of a region P (or $C \in P$) if region C *lies entirely inside* region P. A GRDP region P can create any number of non-overlapping child regions. Specifically, a region P can

1. delegate a child region (say) $C \in P$, or
2. associate various types of information with i) any point inside P or ii) any line wholly inside P or iii) any polygon wholly inside P.

In fact, under a secure, and trustworthy GRDP, each delegation is treated as an atomic transaction T. Only a 'well-formed' transaction results in a transfer of authority of a child-division to a child's authority. On the completion of a delegation transaction, a new association $\langle C, U_c \rangle$ is instantiated and saved as a state change in the delegation system. Once a parent's authority over a child-division has been successfully delegated to a child's entity, the parent entity must not be able to delegate to the other entities. The idea of preventing double delegation is similar to avoiding transfer of the same monetary value to two different recipients through an online remittance service. In an online remittance service, a central trusted institution like Bank or Credit Card Company keeps track of every unit spent from an account to avoid double-spending of a value. On the contrary, a revolutionizing technology such as Bitcoin totally obviated the need of trusted institutions like Banks while also avoiding double spending problem of monetary transactions [46, 70]. The technology behind it is a peer-to-peer, broadcast network operating on an open consensus-based protocol where a double-spending is avoided by having a set of computing nodes

verify a transaction against a complete set of historical transactions stored in an append-only ledger called blockchain [46, 12].

On the other hand, the proposed regional delegation process is similar to a DNS where a name query response is authoritative and uniquely established by delegated zone servers. This protocol, however, can be built on the premise of two different system design models: i) conventional centralized, two-party, prover-verifier model and, ii) consensus-based state transition model. In the following section, we briefly describe how each of these models operate to meet a delegation purpose.

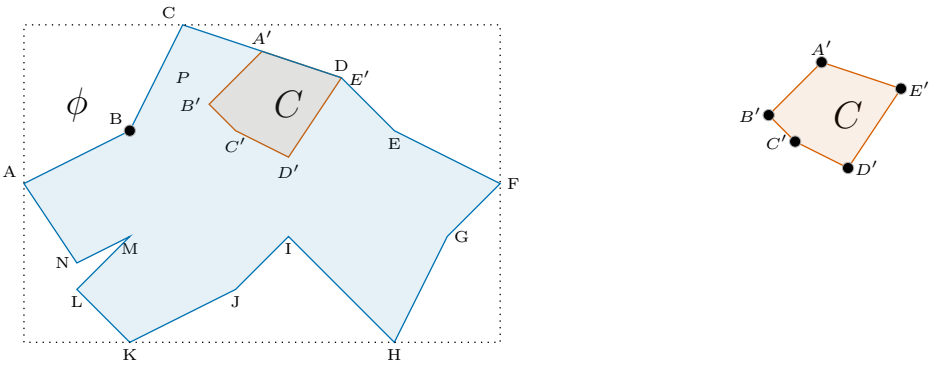

Figure 5.1

A parent entity U_p draws a partition C in the parent region P over her control to transfer the authority over the division to child entity U_c.

5.3 Conventional Delegation Workflow

A conventional delegation model is a centralized two-party communication protocol. For example, the first party is a map server M_s and the second party is a client entity U_c. The map server maintains a complete spatial map, and is respon-

sible to respond to clients' queries. It also performs map modifying operations on the current map. The security of such s system depends on trust on the centralized map service–its processes, and the platform. However, a single platform cannot be entirely trusted upon its outputs. There are several reasons behind it. Firstly, an exhaustive verification of a complex sub-system is virtually impossible. Secondly, a centralized system is highly prone to single point of failure due to malicious attack or technical issues. Hence, in a centralized system, data integrity, and service availability is expensive to realize. An alternative computing model based on distributed consensus protocol and tamper-proof database known as blockchain is deemed suitable for a spatial delegation purpose. The backbone of such a protocol is network consensus-based broadcast blockchain network discussed earlier in section 2.5.2.

5.4 Blockchain-based Delegation Workflow

A blockchain network-based protocol for delegation maintains the dynamic integrity of practically unlimited geo-spatial data items stored in a distributed database. A broadcast network feeds on and validates specific transactions before committing state changes to an append-only ledger called a blockchain. Network nodes are efficient in maintaining data integrity and verifying transactions by storing and referring minimal data, critical data meta-data that summarizes the dynamic state of the delegation system. For instance, such data elements can be cryptographic hashes of data records, an accumulated hash of series of data records or the root of a Merkle hash tree [44] of data records, among others.

The proposed blockchain-based trustworthy delegation protocol, also known as Geographic Regional Delegation Protocol (GRDP), originates from the following robust assumptions.

1. The blockchain network verifies the integrity and correctness of delegation transactions.
2. Blockchain ledger is a distributed append-only ledger, that is a tamper-proof record of states that can be publicly verified by any entities.

The protocol, additionally, makes the following assumptions related to a spatial map.

1. The polygonal region P is a non-intersecting, closed 2D polygonal region bounded by an ordered sequence of boundary segments.
2. The delegated child-division inside P is also a non-intersecting, closed 2D polygonal region.

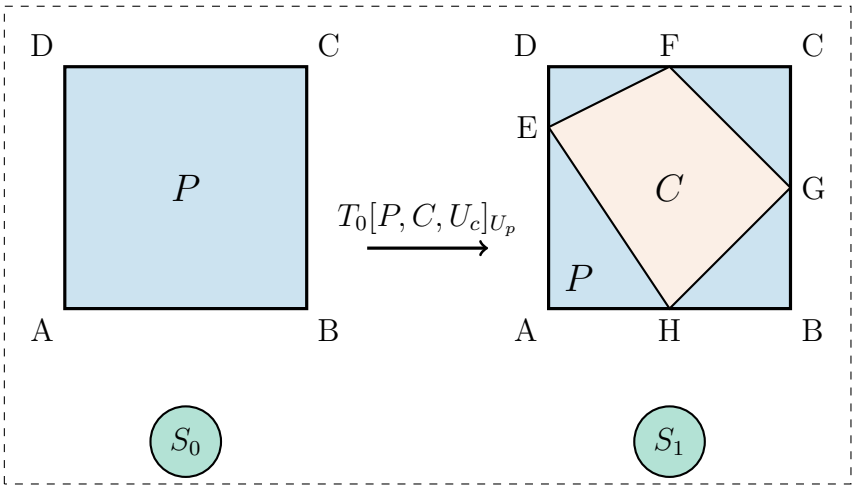

Figure 5.2

A regional delegation is a transaction (T_0) that permits an owner (U_p) to divide a region (P) (owned by U_p) into child-region (C) to transfer the authority over the child region to a child authority (U_c).

Figure 5 depicts a simplified work-flow of a regional delegation process. An entity U_i has authority over a rectangular region $P : ADCBA$. A state S_0 is reached after chain of trustworthy execution of transactions that establishes relationship between U_p and the region P. Consider that U_p wishes to draw partition segments $\{MN, MP, PO, ON\}$ to construct a new child-division C inside the region P. With S_0 as the current state of the system, U_p instantiate a high-level delegation transaction

$$T_0[P, C, U_c]_{U_p},$$

where U_p broadcast digitally signed transaction T_0 over the blockchain network to delegate authority over child-division C to the child entity identified by U_c. The blockchain network's incentivized users (nodes) compete first to validate the transaction T_1, and communicate to reach a consensus on the next state S_1 of the delegation system. The verified transaction and new state are committed to an append-only ledger maintained by each of the network nodes. The new state S_1 is a piece of information that uniquely establishes verifiable, authenticated, relationship between child-division C and child entity U_c. The same procedure is followed by U_c to delegate a child-division C' in C to a third-level child entity $U_{c'}$, and so on.

In the following sections, we identify sub-processes, data objects, pre-conditions, and post-conditions of constituent sub-processes that constitute an AGDP protocol.

5.5 Sub-processes in a Blockchain-based Delegation Process

A delegation process can be divided broadly into two sub-processes $\mathcal{D}_0$ and $\mathcal{D}_1$.

1. SQDM Process $\mathcal{D}_0$: Input to this process is a set of regions $P_0, P_1, ...P_n$. It follows a secure and trustworthy SQDM protocol discussed in chapter 4 to transform a 2D spatial map to a 2D representation known as an SQDM. An SQDM P_{SQDM} is maintained by each of the mining nodes in a blockchain network, and the state corresponding to an SQDM is stored in a ledger.

2. Delegation Process $\mathcal{D}_1$: Input to this process is a sub-division C inside a region $P_i \in \{P_0, P_1, ..., P_n\}$, an SQDM for a polygon P_i produced by process $\mathcal{D}_0$; a reference to a delegating authority U_{p_i}; and a reference to an entity U_c to which delegation is transferred. This process is triggered by a delegation transaction T signed by an authority U_{p_i}. It applies system-specific validation rules to the transaction, and commit meta-data, and the state changes into an append-only distributed ledger.

Shortly, we notice that a delegation process $\mathcal{D}$ works on two sets of inputs:

i) an SQDM for a closed polygonal region $P_i \in \{P_0, P_1, ...P_n\}$ in a 2D plane. In the SQDM, a set $\mathcal{B}$ of segment annotated bounding blocks (BBs) of different types $\{0, 1, 2, .., 14\}$ are maintained as leaves of an OMT $\mathcal{T}_{\mathcal{B}}$, and the dynamic value $\sigma_{\mathcal{B}}$ is stored as a single commitment to every BB in the collection.

ii) a closed child division C drawn inside the region P_i, which is set of consecutive boundary points $\{p_0, p_1, .., p_n, p_0\}$ in CCW. Two consecutive points p and p' is a segment that separates two a 2-D plane into two regions–the one above, ρ_a and the other below, ρ_b. Syntactically, a segment pp' connecting two endpoints $A(x_i, y_i)$ and $B(x_j, y_j)$, and separating two regions ρ_a and ρ_b is defined as:

$$pp'/AB = (x_i, x_j, y_i, y_j, \rho_a, \rho_b) \tag{5.2}$$

A 1-D OMT $\mathcal{T}_c$ with dynamic root σ_c is maintained to track the integrity of the child-division C due delegated. Each leaf of the tree is defined as:

$$(p = n \parallel (x, y), p', v = (x, y))$$

where, n is the index of a point (x, y) in the boundary-points in CCW of the sub-division, and p' is the index of the next point following p'. We use shorthands $pp'(\rho_a)$ or $AB(\rho_a)$, and $pp'(\rho_b)$ or $pp'(\rho_b)$ to refer the regions just above and below a segment pp' or AB.

In the next section, we discuss the general conditions required for a successful delegation transaction.

5.5.1 Validation Criteria of a Delegation Transaction

A delegation transaction $T_t[P, C, S_t, U_c]_{U_p}$ is well-formed if all of the following preconditions (also known as validation rules) are implicitly or explicitly established to be true.

1. T_t is digitally signed by a valid authority U_p, and the signature of which is validated by a procedure VERIFY(U_p, T_t).
2. U_p has an authority over the region P.
3. Child-division C is a sequence of partition segments that forms a closed polygonal region.
4. The child-division C is a simple polygon. Any self-intersecting polygonal child-division is invalidated by process D_1.

   As shown in Figure 5.3, child-division C is a self-intersecting, which forms two connected components. As we walk counter-clockwise along the boundary of d_1 from F to E, a point p' falls interior to the child polygon C. Paradoxically, the same point falls exterior to the polygon C as we walk from A to G. Thus the ambiguity in assigning a point to a region in a self-intersecting polygon leads to invalidate basic assumption that a single line segment divides a plane into two regions.

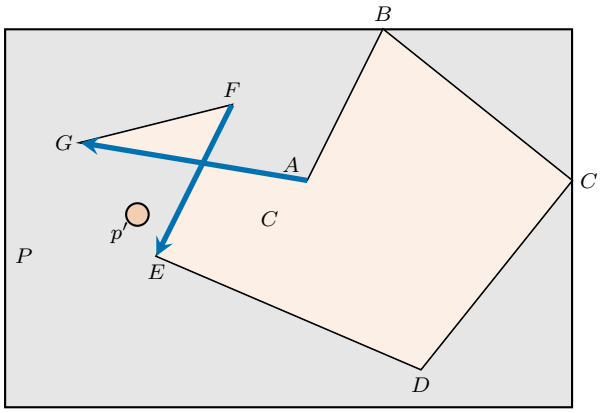

Figure 5.3

Intersecting polygon is invalid for an SQDM process due to ambiguity in assigning a point to an interior to a sub-division.

5. Each segment $s'_k \in C$ must be contained in/on the polygon P.

   Figure 5.4 shows valid and invalid partition segments drawn by entity U_p who has authority over region P in an SQDM.

   As shown in Figure 5.4 (a), child-division $d_0 = \{EFG..JE\}$ is drawn in a Type 0 block in an SQDM. Segment IJ is a legal segment on the boundary of the region P. Similarly, segment FG, GH, HI touch the border of the region R are all legitimate. Any segment completely inside the block is also a valid partition segment. However, because EF and JE cross out the block, they are illegitimate partition segments. Any illegal segment in a sub-division renders that sub-division to be completely illegitimate partition.

   In Figure 5.4 (b), segments $\{H'I'\} \in d_1$ are invalid partion segments because they cross diagonal of a Type 1 or bBB block; segments $\{L'G', G'J', J'K', K'L'\} \in d_2$ are valid partition segments for the lower half of the block.

   As shown in Figure 5.4 (c) segments $\{L'K', K'J'\} \in d_4$ are invalid because they cross third subdivision $R' \notin U_i$ just below segment $A'C''$ in $TypeII$ or (rBB) block; however, division $L'J'_1J'_2J'G'L'$ is a valid partition for the region below main diagonal segment $A'C'$ and above segment $A'C''$ in a typical Type 7 block.

   A procedure BelongTo(s'_k, C_j, P) returns TRUE if a segment $s_k \in C_j$ is a legal child partition segment for a parent region $P \in U_p$. This procedure must return TRUE for all segments $s'_k \in C_j$ for child division C_j to be a legitimate child-division inside parent region P.

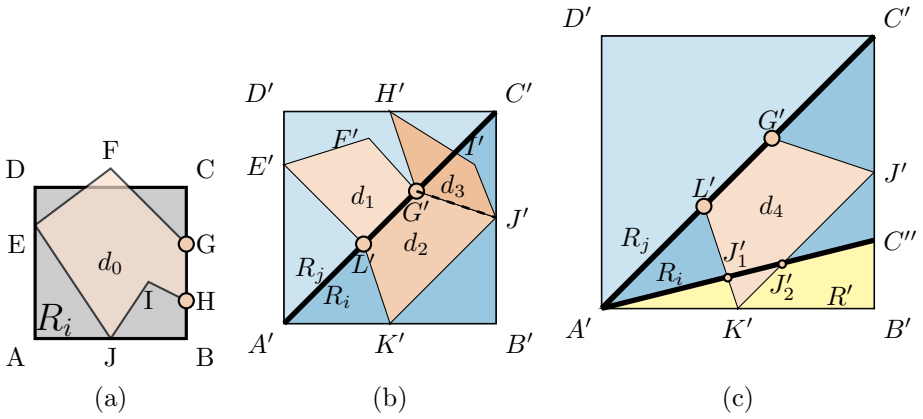

Figure 5.4

Invalid partition segments for three types of BB in an SQDM. (a) shows the partition segments in Type 0 block; (b) shows child-division in a typical Type 1 block; and (c) shows sub-division in a typical Type 7 block.

5.5.2 Simplified Illustration of a Delegation Protocol

Consider a simplified polygon $ANM...CBA$, as shown in Figure 5.5 (i). The polygon is bounded by a sequence of boundary points $\{A, N, M, ..C, B, A\}$ in CCW. Assume that the region interior to the polygon is P and the exterior to the polygon is region ϕ. Let us assume that an entity U_p has an authority over the region P, encoded by a mapping $\langle U_p, P \rangle$.

First of all, we execute an SQDM protocol (as discussed in Chapter 4) to produce a 2D block representation of the input 2D map. A successful execution of the protocol will result in an SQDM, as shown in Figure 4.2 (Right). It is an assembly of three types of bounding blocks BB covering the regions P_0 and P_1.

- **Type-0**
- **Type in** $\{1, 2, 3..6\}$
- **Type in** $\{7, 8, 9...14\}$

Assume that an authority U_i draws a sequence of partition segments to construct a closed sub-division C:

$$C_j = \{A', B', C', D', D, A'\}$$

inside the region P. The authority wishes to delegate the child-division (as depicted in Figure 4.7 (ii)) to second entity U_c. For simplicity, we assume that all the boundary points in the sub-division are given in a counter-clockwise of their occurrence in the sub-division C.

From now on, we call the region P as parent region and its boundary segments as *parent* segment; the sub-division as *child*-division and its boundary segments as

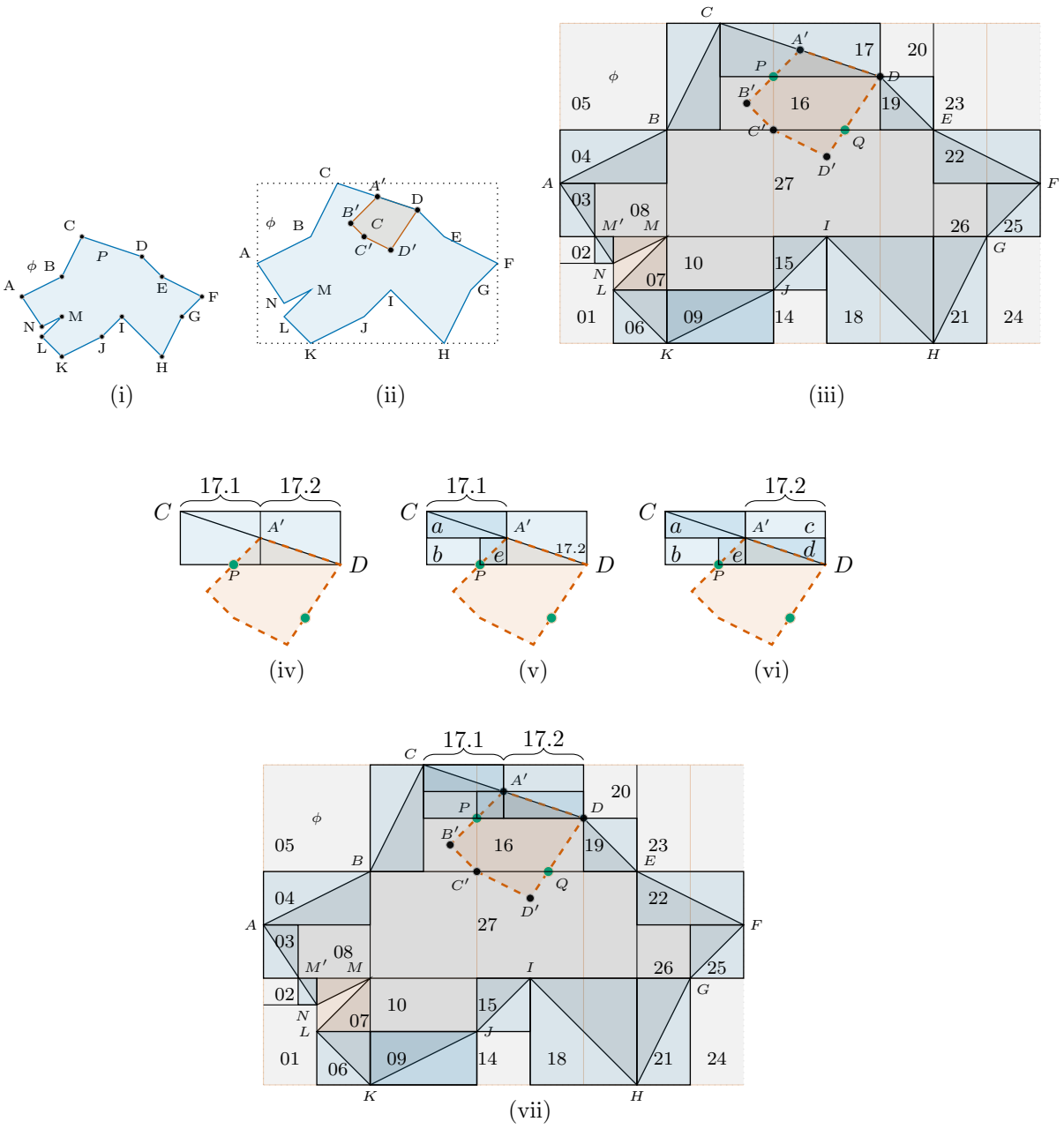

Figure 5.5

(i) Input polygon $ANM...CBA$ has interior region P (under control of authority U_p), and the exterior region ϕ. (iii) SQDM for P is a 2D-OMT $\mathcal{T}^p_{\mathcal{B}}$ with root $\sigma^c_{\mathcal{B}}$. (ii)U_i draws a child-division $C = A'B'C'..E'A'$ inside P to delegate it to the second entity U_c. SQDM for C is a 2D-OMT $\mathcal{T}^p_{\mathcal{B}}$ with root $\sigma^p_{\mathcal{B}}$. (iv)-(vi) Partition segment $A'B'$ is rarified to fit inside an existing BB 17, which is again split to fit the rarefied fragment $A'P$ of $A'B'$ completely. (vii) Rarefied segments in child-division C are inducted into the parent SQDM to complete the delegation process.

child segments; the SQDM for the region P as a parent-SQDM and the SQDM for the sub-division as a child-SQDM; the BBs in parent SQDM as parent-BBs; the boundary point OMT for the parent and child polygonal regions as parent boundary point OMT and child boundary point OMT, respectively.

At time t, U_p instantiates a transaction T_t as: $[P, C, U_c]_{U_p}$. The transaction triggers delegation process $\mathcal{D}_1$.

1. Populates a boundary-point OMT $\mathcal{T}_c$ with static root σ_c for the child division C.
2. Invokes macro-transaction to construct a child SQDM $\mathcal{T}_{Bc}$ with root σ_{Bc} for child division represented by $\mathcal{T}_c$. This process is described in Chapter 4. Specifically, we have to validate the following two requirements for a child-division.
   (a) A child division is a simple polygon: The ability to successfully map all the boundary segments to BBs in an SQDM, in the CCW order of points in a boundary-point OMT for the polygon is sufficient to determine that no two segments (other than adjacent) intersect with each other. As two crossing segments cannot be mapped to the same BB, whenever we encounter that two segments being assigned to the same BB intersect, we identify a non-simple polygon. Alternatively, if two segments are divided at the same point, we also identify intersected segments. To support it, we maintain counts of the boundary points and the division points for a polygon. Whenever the count of boundary points inserted becomes greater than one, then we identify an intersecting polygon.
   (b) Child division is a closed loop: Secondly, it proves that the polygon is complete – by recording the first point and ensuring that the last point is the same as the first. Thirdly, we can easily attribute the appropriate region identifier (1 or ϕ) to each line segment incorporated in a BB. Most importantly, on completion of the mapping, a query regarding any point (x, y) can be answered by examining a single BB in which (x, y) falls – by determining where the point (x, y) falls inside the BB in relationship to the line(s) inside the BB.
3. Rarefy child segments: This process divides child segments along either horizontal or vertical axes such that the splits completely fit into a BB in a delegating SQDM. Here in the particular case (see Figure 5.5), child segment $A'B'$ is split along horizontal axis $y = D_x$ (at green filled point on $A'B'$) so that the splits

$A'P'$ and $P'B'$ fits well into two different BBs **17** and **16** respectively. This is achieved by invoking μ-transaction

$$\mathsf{AddPoint}(\mathsf{A'}, \mathsf{P}, \mathsf{x} = \mathsf{D_x})$$

Similarly, the partition segment $D'D$ is split along the horizontal axis through $y = E_x$ to produce splits $D'Q$ and QD which fits inside BBs 27 and 16 respectively, by invoking

$$\mathsf{AddPoint}(\mathsf{D}, \mathsf{D'}, \mathsf{x} = \mathsf{E_x})$$

Child segment $A'D, B'C, C'D'$ do not suffer any split because they fall completely inside BB **17, 16** and **27** respectively. In general, child segments are split by invoking μ-transaction

$$\mathsf{AddPoint}(\mathsf{p}, \mathsf{p'}, [\mathsf{x'}, \mathsf{y'}]),$$

where, $[x, y]$ is the axis along which the segment is split. Particularly, the split axis is either of 4 sides of a BB in the input SQDM, where segment pp' overlaps geometrically.

Let us collect these splits in a dictionary $L_{rc} = \{A', P, B', C', D', Q, D, A'\}$.

4. Map child segments in L_{rc}: Once child segments rarefied so that they fit completely inside existing BBs in the parent SQDM, they are sequentially mapped to the parent BBs.

   For example, child segment $\{A'P, A'D\}$ are mapped to parent BB 17; $\{B'P, B'C', QD\}$ are mapped to the parent BB **16**, and $\{C'D', D'Q\}$ are mapped to BB **27.**

   However, mapping a child segment pp' to a parent BB is valid if the segment(s) becomes:

   i) a diagonal of the parent BB

   ii) a support segment of a diagonal of the BB (one endpoint is at the corner of the BB and other is on the opposite side of the BB)

   iii) one of the 4 sides of the parent BB with segment (parent or child) mapped previously.

   For instance, segment PA' in BB **17** fails condition i), ii) and iii); segment $A'D$ fails condition ii). This calls for invoking a μ-transaction

   $$\mathsf{SplitBB1}(\mathsf{pp'}, \mathsf{BB})$$

   to further split the parent BB so that a child segment can be correctly mapped.

   Consider a BB **17** along parent boundary line CD and child segment PA' from map in Figure 5.5(c). The following μ-transactions are invoked to additionally split parent BB **17.1** into constituent parent BBs.

i) $\{17.1, 17.2\} \leftarrow \mathsf{SplitBB1}(17, \mathsf{x} = \mathsf{A}'_{\mathsf{x}})$ //produces BBs 17.1 and 17.2

ii) $\{a, b\} \leftarrow \mathsf{SplitBB1}(17.1, \mathsf{y} = \mathsf{A}'_{\mathsf{y}})$ // produces BBs 'a' and 'b'

ii) $\{b.1, e\} \leftarrow \mathsf{SplitBB1}(\mathsf{b}, \mathsf{y} = \mathsf{A}'_{\mathsf{y}})$ //produces BBs 'b.1' and 'e'

By now, we are ready to map child segments $A'P$ to a parent BB e. Similarly, parent BBs **27** and **16** are split to map partition segments $B'C'$, $C'D'$, and QD.

5.5.3 Region OMT

The region OMT is also a collection of key-value pairs; a pair $\{R, v_R\}$ corresponds to a region R with parameters v_R. Specifically, the parameters (in v_R) associated with region R include:

α	Context
ξ_{ps}	root of boundary points collection
ξ_{bs}	root of BB collection (pre-delegation)
ξ_{bd}	root of BB collection (includes delegated regions)
ξ_r	root of "parent and children" key-value collection

From now on we shall denote the parameters of R as $R.\xi_{ps}, R.\xi_{bs}$, etc. The context α can be anything that this map represents in the real-world. For instance, the map for context "State," "City," "Congressional District," "Gas Stations," "Personal Parcels," and so on.

The 1-D OMT root ξ_{ps} corresponds to a boundary-point OMT $\mathcal{T}_p$ collecting boundary-points in CCW for a region R. The 2-D OMT root ξ_{bs} corresponds to a static map of the region R (before delegation of sub-regions) represented by $\mathcal{T}_B$; ξ_{bd}

corresponds to a dynamic map ${\mathcal{T}'}_B$ altered by delegations. Leaves in both collections are 5-tuples of the form (x_l, y_l, x_h, y_h, v).

The key-value pair collection with root ξ_r has one entry corresponding to the parent, and one for every child region; a key-value pair $\{Q, q_c\}$ in this collection indicates that region Q with region code $q_c > 1$ is a delegated child; if $q_c = 0$ the region is the parent. The region code 1 is reserved for itself (undelegated regions).

If $R.\xi_{bs} \neq 0$, the implication is the successful completion of the map construction process. Such a region is said to be in a *non-authoritative* state until $R.\alpha$ remains 0.

In the authoritative state, $R.\alpha \neq 0$ (R has been assigned a context), and $R.\xi_r \neq 0$ (as R must have a parent).

5.6 Surrendering/Rolling-back a Delegation

A process by which a receipt of authority over a sub-division in a 2-D space is returned back to the immediate authority that delegated the sub-division is defined as surrendering (or rolling-back) a delegation transaction. It is similar to refunding an online transaction. The need for rolling-back a delegation in the context of a spatial region comes with few reasons:

- an error in construction of delegated sub-division,
- an erroneous delegation to wrong authority
- a service failure or due to legal interruption, among others.

A secure protocol for rolling-back a delegation is expected to totally preserve the area of delegated region after the rolling-back.

A transaction RollBack (R, q_r, P, v, p, p_n) initiates rollback of region R delegation by parent P. Before a region R delegation can be rolled back, delegations of all regions inside R should have been rolled back. In other words,

1. $R.\xi_r$ should be the root of an OMT with a single key-value pair, $\{P, 0\}$, corresponding to the parent P;
2. $\{R, q_r\} \in P.\xi_r$;
3. The context OMT should have a key-value pair $\{\alpha, \xi_d \parallel \xi_c\}$ where $\{R, P\} \in \xi_d$, $\{R, 0\} \in \xi_c$, and $\{P, n\} \in \xi_c$

After the delegation is rolled back $R.\xi_r = 0$, $R.\alpha = 0$, $R.\xi_{bd} = R.\xi_{bs}$, $R \notin P.\xi_r$, $R \notin \xi_d$, $R \notin \xi_c$ and $\{P, n-1\} \in \xi_c$. This macro-transaction conveys a sequence of micro-transactions Remline (p, p', q_r) to remove a boundary line (p, p') (not shared with the parent) and modify a region identifier associated with each line from q_r to 1. The network nodes must achieve consensus on an OMT root r_b and start/last-seen points p_s/p_l after executing the sequence of micro-transactions. Only if $((p, p_n) \in R.\xi_{ps}) \wedge (p > p_n)$, r_b is initialized to $P.\xi_{bd}$, and $p_s = p_l = p_n$.

5.7 Related Works, Results, and Conclusions

Considering that a state authority delegates county-level boundary to county authority, states to county delegation was performed to evaluate the overhead of the micro-transactions. For instance, Mississippi state is delegated to constituent counties such as Yazoo, Noxubee, etc. The delegation process begins with independent parent-level SQDM for the state and child-level SQDM for a county. For instance, to delegate Yazoo county with 717 boundary segments by Mississippi state SQDM (with 7919 BBs), child boundary segments must be **rarefied** 212 times, producing a total of 929

(717+212) child segments. These child segments fall into the total of 74 parent-BBs. These 74 parent BBs were split 2518 times to accommodate (929) child boundary segments. After mapping the boundary segments (Figure 5.6 bottom), the total number of BBs in the parent SQDM was 10363 (6682 clear BBs, 3661 blue/green, and 20 red).

The GRDP protocol provides the following key merits in relation to spatial authority transfer:

1. It supports fail-safe decentralized delegation services.
2. It obviate a trusted-third party authority for a delegation of authority over two a dimensional space.
3. It supports the self-governing system of operating nodes.

Confidence in the attribution of ownership/control of geographic (regions (or co-ordinates) opens avenues for a wide range of trustworthy services. For instance, in the current time, anyone seeking to provide an (on-line) service is required to

1. Purchase an Internet domain names such as *abc.com; xyz.org*, etc.) from a domain registrar.
2. Promote the domain name, with the hope that potential clients will remember the name when they need the service.
3. Provide DNS information regarding the name (in the form DNS records) to enable users to contact for purchasing a service.

However, if geographic coordinates can be delegated from one entity to another entity, business or service providers do not need to purchase and promote their Internet domain names. For instance, they already own a geographic location that can uniquely identify their business or services. Potential customers do not need to

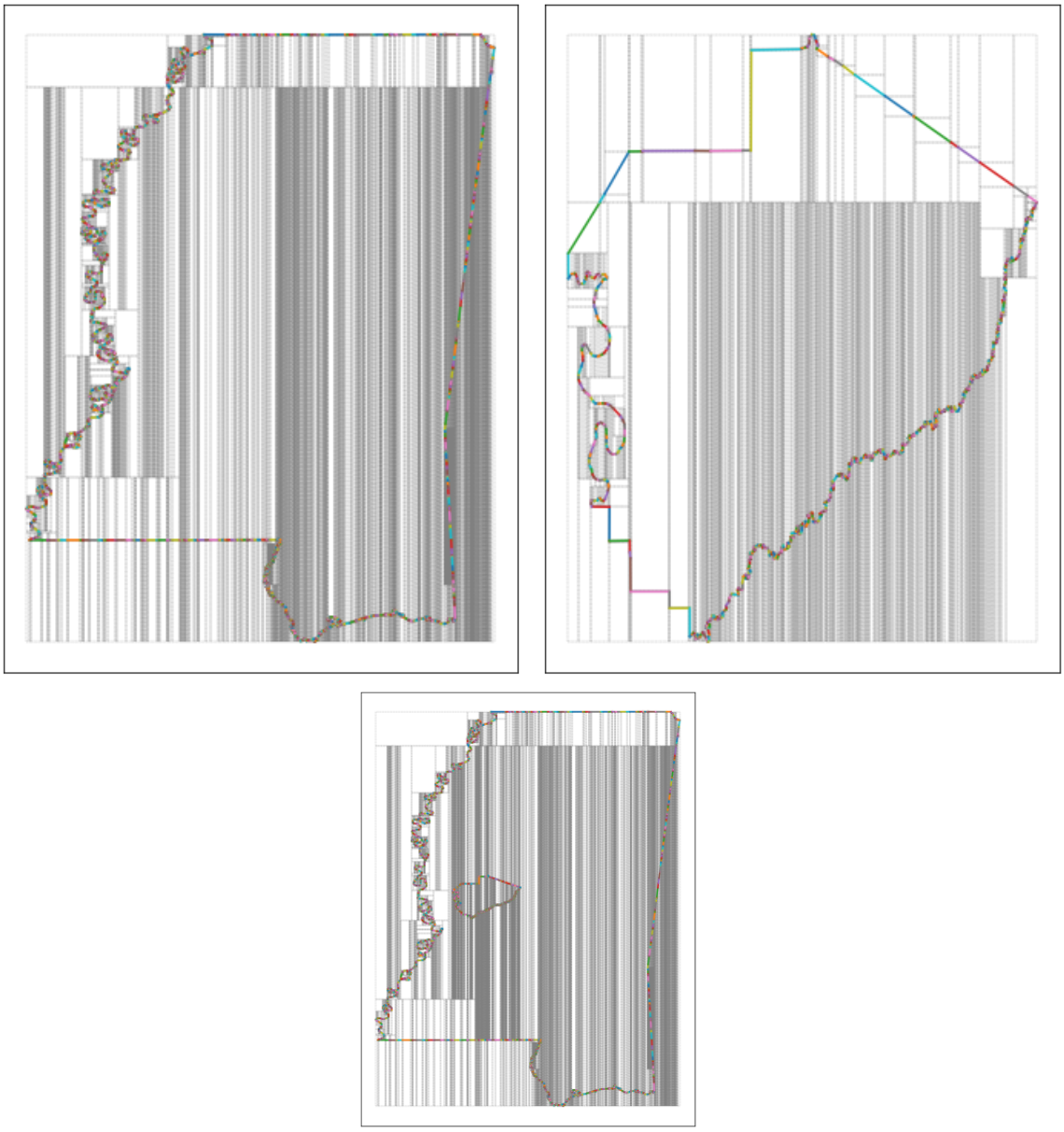

Figure 5.6

Top (left): SQDM for MS state. Top (right): SQDM for Yazoo county. Bottom: After the delegation of Yazoo county by MS.

recall domain names of the services/business and also do not have to search them by a location. With an authorized delegation service, any customer that look for a service can simply search by a location to get an authenticated, and unbiased list of service provider around the location by using a simple query such as '*n-closest context-specific feature around (x,y).*'

CHAPTER 6

BLOCKCHAIN-BASED REDISTRICTING PROTOCOL

> Either you repeat the same conventional doctrines everybody is saying, or else you say something true, and it will sound like it's from Neptune
>
> *Satoshi Nakamoto*

Trust in the integrity of processes for congressional redistricting is crucial for the smooth functioning of democracies. Achieving universal consensus on the fairness and unbiasedness of district plans is essential. In this chapter, we discuss a blockchain-based redistricting protocol (BREP) to automate the task of redistricting. It is expected to enhance public participation in a redistricting process and bolster transparency in the process outcome. BREP is designed to choose the "best" of any number of independent redistricting plans, based on agreed-upon metrics like iso-perimetric ratio, area moment, population moment, among other metric that measures the compactness of the constructed districts. Other constraints, such as contiguity of the geometry and equal-population, are also evaluated.

6.1 Introduction and Motivation

In the context of elections, redistricting (or only districting) is a task of decomposing a geographical region into electoral sub-regions called districts such that

each sub-region meets a similarity criteria such as population, geographic contiguity, geometric compactness, among others [22]. Every ten years, congressional redistricting is performed to incorporate population changes across the United States. Basic principles of a congressional redistricting in the USA include [3]:

1. Each new district must have a nearly equal population.
2. Each district must avoid political biases.
3. Each district must form a geographically contiguous and geometrically compact geographic region.

The redistricting principles, however, may differ among states. A congressional redistricting is usually performed by a commission of political representatives or independent experts. Redistricting by a politically biased committee may lead to what is known as gerrymandering [3]. A gerrymandered redistricting process produces districts that give an advantage to one party over another [3, 19] in elections. The idea of using computer algorithms to automate redistricting to produce a fair district plan is appealing to scholars in political science, law, and computational sciences. However, the trust in the computer platform and the algorithm are controversial, for very good reasons. The integrity of the computing platform, data or algorithms may have been compromised (or attacked) at numerous points. Sophisticated computer algorithms might be misused to produce appealing but subtly, politically biased districts [3].

Moreover, most of the redistricting algorithms or software use some heuristics algorithms such as K-Means minimization heuristic, shortest split-line algorithm, and ad-hoc heuristic, among others [3]. These algorithms neither yield precise and correct solutions nor provide a probability of correctness/error of the solution. Redistricting

operations, in essence, seek to create optimally compact, contiguous, near-equal population districts, thereby optimizing some cost, making it an NP-hard problem [4]. It is therefore easy to understand that present computing platform and algorithm(s) are not sufficient to produce or identify unbiased districts. We purpose a trustworthy and secure blockchain-based redistricting evaluation protocol (BREP) for choosing the "best" districting plan from potentially thousands of proposed districting plans, based upon agreed-upon compactness measures such as iso-perimetric ratio, area moment, population moment, etc. The motivations for the proposed approach are as follows:

1. A redistricting problem is an exhaustive search problem that cannot guarantee optimality.
2. A computerized or automated redistricting method cannot be trusted to generate a fair redistricting plan.
3. An ability to independent experts to participate in a redistricting process can enhance transparency, competitiveness, and consensus in the fairness of districting tasks.

Having discussed the rationale behind purposing BREP, we formalize the protocol in the following section.

6.2 Overview: Blockchain-based Redistricting Protocol (BREP)

A geographic region (such as a state of the USA) is a (political) map containing non-overlapping boundaries called census blocs. We group the census blocs/blocks together to construct different contiguous sub-regions. These sub-regions are called as congressional districts. For instance, the state of Texas is constituted with 914231

census blocks. These blocks are grouped into 36 different political sub-divisions called congressional districts. Each district is a contiguous geographic region.

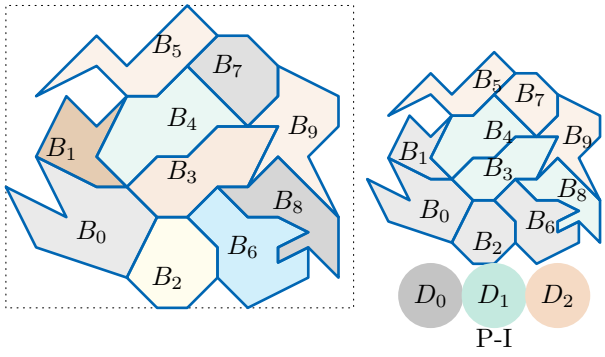

Figure 6.1

Left: a region R contains ten census blocks $(B_0, B_1, ..B_9)$ allocated for redistricting into three districts D_0, D_1, and D_2. Right: P-I, where the ten blocks are grouped to form 3 sub-regions, (say) congressional districts as depicted by gray, green, and red shade areas.

Consider a geographic region R, as shown in Figure 6.1. It consists of a set of 2-D sub-divisions, viz., census blocks $\{B_1, B_2, .., B_n\}$. Each census block $B_i \in R$ is geography bounded by a set of boundary points $p_0, p_1, ..$ in counter-clockwise order (CCW). Each sub-division B_i is also associated with a set of p static attributes $\mathscr{A}_i = \{a_0^i, a_1^i ... a_p^i\}$. For instance, attributes of a census block can be a scalar measure of an area enclosed, count of the population residing in the block, among others. The region R is divided into q sub-divisions called congressional districts, $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2 .. \mathcal{D}_q\}$, where a set of contiguous census blocks creates each district.

A redistricting plan is a mapping $\mathcal{G} : R \rightarrow \mathcal{D}$ that maps each $B_i \in R$ uniquely to one of the distinct districts $\mathcal{D}_j \in \mathcal{D}$. Furthermore, such mapping must attempt to optimize k measurable constraints, $\mathcal{C} = \{c_1, c_2...c_k\}$. For instance, a plan may impose measurable constraints such as equal-population, compactness, convexity, and proximity of the districts constructed. Given a set of N district plans $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, ...\mathcal{G}_N\}$, BREP is a protocol for a trustworthy computation of a 'goodness' measure m_i for each of the mapping $\mathcal{G}_i$. Let us take an example of an online auction process to explain the operations of BREP protocol further.

6.2.1 An Analogy to an Online Auction Process

A seller auctions a product O for an online sale. Potential n customers bid their intended payment amounts $\mathbb{X} = \{x_0, x_1, ..\}$ online. As the auction expires, the highest bidder $C_j \in \mathbb{X}$, bidding maximum price $x_{max} = max\{\mathbb{X}\}$, is announced as a winning bidder. For a fair and impartial result of a bidding process, a secure and trustworthy system must have control over the auction process and related bidding data assets such as customer's account, bidding amounts, etc. A mutually trusted party to select the highest bidder is necessary to uphold the trust in the entire auction system. Similar to an online bidding process, a BREP protocol begins by posting a valid problem P, that defines a state-wide (or nation or city-wide) redistricting problem in a public blockchain infrastructure. The blockchain infrastructure consists of possibly hundreds of independent computing nodes, often called verifiers. The

verifiers (also known as incentivized users) verify the integrity and correctness of the problem instance before finally committing to a public blockchain ledger.

Independent sources (users) can propose their districting plan (or redistricting solution) for a specific valid problem committed on a blockchain network. Such independent sources can be an individual, an expert group, a firm, a computing node or a system of resourceful computing nodes, and so on. A source of a redistricting solution $\mathcal{G}_j$ to a redistricting problem P broadcasts the solution over a public blockchain. According to the BREP protocol, each redistricting solution must be associated with a metric value that is a measure of 'goodness' (higher compactness measure, near-equal population, and contiguous geographic boundary, among others) of redistricting solution. Additionally, the measure and certain monetary stake (e.g., a digital currency such as Bitcoin). Once a bunch of the redistricting solutions is received by the blockchain network, the "best" solution is selected by the network depending upon the claimed 'goodness' measures of the solutions. The best solution is again validated by the incentivized nodes (users) of the blockchain network. If such a solution fails the validation criteria defined for the corresponding redistricting problem, the stake associated with the solution is forfeited by the blockchain network. A valid redistricting solution is declared as the winner of the redistricting process and is rewarded based on predefined reward and penalty policies for the problem. The provision of reward and penalty ensures the submission of correct solutions and eventually selection of the "best" redistricting plan.

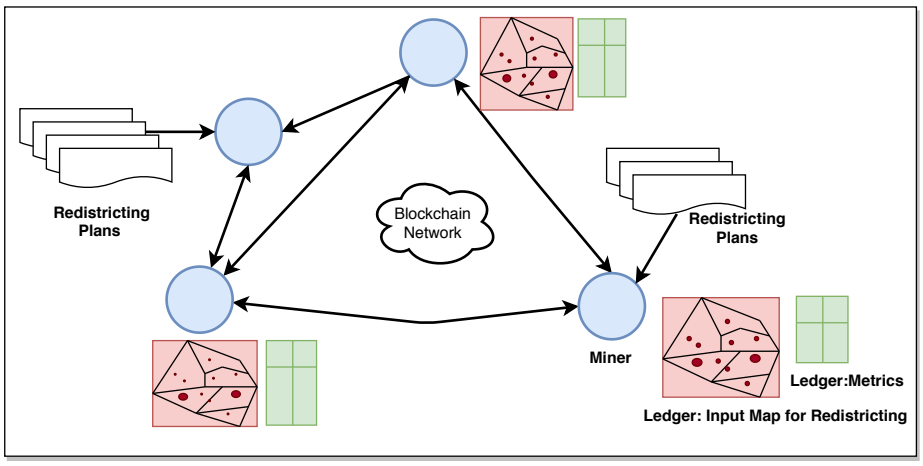

Figure 6.2

In a BREP, a map redistricting problem is defined and broadcast over a blockchain network. A set of independent computing nodes submits a set of redistricting plans to the blockchain network. The potential plans are evaluated by executing transactions corresponding to standard metrics. The evaluation results are validated in blockchain and finally committed to a ledger.

Figure 6.2 depicts a simplified framework of a BREP protocol. A state authority publishes data assets required for a redistricting over a public blockchain network operating on a BREP protocol. The data assets are verified before they are committed to an append-only ledger in the blockchain network. Identified non-state authorities post potential district solutions (plans) for the problem into the network. The incentivized nodes (users) in the network agree upon the "best" solutions. They eventually validate it to determine the best redistricting solution.

In short, fairness of the bidding process may call for a trusted third party to select the highest bidder. The goal of BREP is to avoid the need for a trusted third party to select an optimal redistricting plan. BREP employs a transaction-based state machine model for achieving consensus on the outcome of the process state.

Specifically, BREP relies on a consensus-based blockchain broadcast network (BN) [46, 12].

6.2.2 Security Assumptions

The security and trust of a BREP stem from the following assumptions:

1. A blockchain network verifies and maintains the total integrity of input data required for redistricting. Input data set in used by BREP consist of
    - geo-spatial information of census blocs (or counties) in a given geographic region,
    - static attributes such as count of population and area/perimeter/centroid of the census blocs,
    - contiguity and compactness requirements
    - policies for redistricting reward and penalty,
    - redistricting plans (solutions), among others.
2. Incentivized computing nodes (also called miners) compete to determine the "best" redistricting plan out of the submitted redistricting solutions. Once it is selected, the network nodes again compete to validate the plan.

6.2.3 Contributions

This chapter makes the following contributions:

- Efficient data structures for low-complexity verification of transactions executed in a BREP.
- Execution policies that a blockchain network nodes to follow to complete the selection of the "best" redistricting solution submitted to the network.
- Open avenues to enhance public participation in a pressing issue of congressional redistricting.

The security and trustworthy of a redistricting evaluation protocol leverages two major components:

- A set of Ordered Merkle Tree (OMT) for ensuring the integrity of redistricting data assets; district plans; and evaluation metrics corresponding to each of the plans, and
- A universal consensus achieved on the integrity of the operations and data in a blockchain network.

6.2.4 Metrics for Evaluating District Plans

Generally, geometric compactness, contiguity, and population equality have been considered for measuring the quality of a redistricting plan. Litany of scholarships are dedicated to measuring a districting plan for a redistricting problem. In cases, heuristic algorithms used to produce a districting plan attempt to minimize or maximize compactness while also producing near-equal population districts. Other greedy algorithms just attempt to produce equal-population districts. However, scholars still debate over complete dependence on a single criterion. We will define some of the frequently recommended compactness measure as reported by [61, 3, 20], to measure the quality of a districting plan. We define several other criteria for a shape d whose area is A_d; perimeter is P_d; the convex hull is H_d; the largest inscribed circle in d is c_d; bounding circle C, and count of population is p_d, and centroid is $c_0(C_x, C_y)$. The shape d consists of N composing shapes called blocks $\{b_0, b_1, .., b_N\}$.

6.2.5 Area, Perimeter and Centroid of a Closed Geometric Figure

Let d be a simple polygon whose n sides are described by Cartesian coordinates $(x_0, y_0), (x_1, y_1), ..., (x_{n-1}, y_{n-1}, (x_0, y_0))$ in counter-clockwise (CCW) along the

perimeter of the polygon. An area, A, of the shape is given by Gauss's area formula [10],

$$A = \frac{1}{2}|\sum_{i=0}^{n-1}(x_i + x_{i+1})(y_{i+1} - y_i)| \tag{6.1}$$

The perimeter (L) and the centroid (C_x, C_y) of the polygon with an area A is given by [9]:

$$L = \sum_{i=0}^{n-1}\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}; \tag{6.2}$$

$$\begin{aligned} C_x &= \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} + x_{i+1} y_i) \\ C_y &= \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} + x_{i+1} y_i) \end{aligned} \tag{6.3}$$

Algorithms for computing A, L, and (C_x, C_y) requires only $\mathcal{O}(n)$ iterations. This permits a low-memory platform to calculate these properties.

Several measures of compactness for a shape d can be defined as below:

1. Iso-Perimetric Ratio (IPQ): It is defined as the ratio of a shape's area to that of a circle of equal perimeter. A higher ratio is desired for a higher compact district. This metric can detect complex boundaries. However, it fails to capture population distribution. Mathematically,

   $$IPQ = 4\pi * A_d / P_d^2; \tag{6.4}$$

   A similar measure known as Schwartzberg ratio equal to $1/\sqrt{(IPQ)}$ can be defined for the shape. However, as cited by [14], this measure is overly sensitive to small changes in the boundary.

2. The moment of area (I): It is a dispersion measure defined as weighted distances ρ_i^2 squared of the composing blocs to a fixed point known as centroid. The weights w_i of the composing blocs can be of area A_{b_i} or population count of p_{b_i}. If weights are measurement of areas, then it is called moment of area (IA), and if weights are measures of population counts, the measure is called moment of population (IP). Though a higher value of the moment is desired, it does not always detect complex boundaries. Mathematically,

$$I = \sum_{b_{i=0} \in d}^{b_n} w_i \rho_i^2 \tag{6.5}$$

3. Convex Hull Area Ratio (HA): It is a ratio of a shape's area to the area of a convex hall for the shape. Instead of an area, it is also possible to define the population hull (HP) as a ratio of shape's population measure shape to population. A convex shape is considered to be less diverse and morphologically, less distorted. Mathematically,

$$HA = A_d / A_{H_d} \tag{6.6}$$

4. Reock Ratio (RR): It is the ratio of the area of a shape to the area of the smallest circle bounding the shape. Mathematically,

$$RR = A_d / A_C \tag{6.7}$$

   a similar ratio can be defined if we take the largest circle c inside the shape. However, it is computationally intensive to determine the largest in-circle for an irregular shape.

5. Mean (MR), Dynamic (DR) or Harmonic (R) Radius: The mean radius is the average value of the radius ρ to the shape's centroid. It is an areal dispersion measure equal to:

$$MR = (2R/3) / \Big((1/A_d) \int_{S_d} \rho \delta A_d \Big); \tag{6.8}$$

   where ρ is a radius integrated over a surface area A_d in shape d. Similarly, dynamic (DR) and harmonic radii (HR) is equivalent to:

$$DR = (R/\sqrt{2}) / \Big((1/A) \int_{S_d} \rho^2 \delta A_d \Big)^{-1/2} \tag{6.9}$$

$$HR = (R/2) / \Big(A / \int_{S_d} \delta A_d / \rho \Big); \tag{6.10}$$

6. Rohrbach Index (RI): It is a measure of an average distance of points in a shape to the shape's perimeter. Normalized with equal area circle, it is given by:

$$RI = (\int_{S_d} d_p \delta A_d) / (\pi R^3 / 3); \tag{6.11}$$

   where d_p is a distance of a point in shape to the shape's perimeter.

Other criteria for redistricting are contiguity of districts, and near-equal population counts among constructed districts.

- District Contiguity or non-fragmentation: This property can be checked by maintaining an adjacency matrix of the census blocks in a state. Each block that falls in a district must be checked if that block's adjacent block is in the district. Figure 6.3 shows a valid and invalid redistricting scenarios.

Algorithm 2: District Contiguity Algorithm

Input : $R = \{b_0, b_1..b_N\}$, a set of blocks (census blocks) in a state R;
$A\langle b_i, \{b_j : b_j \in R \text{ is adjacent to } b_i\}\rangle$–is an adjacency matrix of blocks in R;
$\mathcal{D}$, a districting plan, is a set of k sub-sets of blocks in R.

Output: False if a block b_i in a district $\mathcal{D}_j$ does not have an adjacent block in A that is also in j^{th} subsets D_j.

```
1 for Each b_i IN D_j do
2 |   A_{b_i} := Adj(b_i, A) ;          // Adjacent blocks of block b_i in A
3 |   if A_{b_i} Intersection D_k IS {} then
4 |   |   RETURN FALSE
5 |   end
6 end
7 RETURN TRUE
```

- Population Equality: It is a basic requirement for a districting plan. Algorithms must work to map nearly equal population among created districts. Each census block is associated with a count of population in that block. The total population in a district must be equal to the sum of the population counts for all the census blocks mapped to that district.

  Let $\mathcal{D}_j$ be j^{th} district of a state R, which is composed of $n-$blocks $\{b_0^j, b_1^j, .., b_n^j\}$. Then the population of the district $\mathcal{D}_j$ is given by the sum of the population of the constituent blocks $b_0^j, .., b_n^j$. The variance of the population among the district must be below some defined threshold.

6.3 An Illustration of a State Redistricting Problem

Consider that a public key, U_p associated with an authority responsible for constructing a redistricting region R and announcing the problem in a broadcast blockchain network. Let the contrived region R represents the state constituting of 10 census blocks,

$$R = \{B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9\};$$

allocated for redistricting into three distinct, non-overlapping congressional (say) districts $D = \{D_1, D_2, D_3\}$. Each block B_i is associated with a set of attributes $\{a_0^i, a_1^i, ..\}$. For example, the block's surface area and population count can be denoted by a_0^i, and a_1^i. The geometric attributes such as area and perimeter associated with a geometric figure (such as a polygon) is computed by the network nodes. We also choose to associate non-geometric attributes such as population (p) with each of the blocks in the region. As a requirement for a redistricting problem, the authority U_p is permitted to decide to measure a redistricting plan against $k = 3$ measurable constraints such as $C = \{IPQ, I, HA\}$, with usual meanings as discussed in earlier section 6.2.4.

Starting from now, we refer U_r as a State authority, and her region, R is apportioned for a redistricting. The potential districting plans for the region will be evaluated in a blockchain network that executes a BREP protocol. For notational convenience, we use $U_p.R$, $R.D$, $R.C$, $B_i.a^i$, $B_i.p$, etc. to access parameters/properties

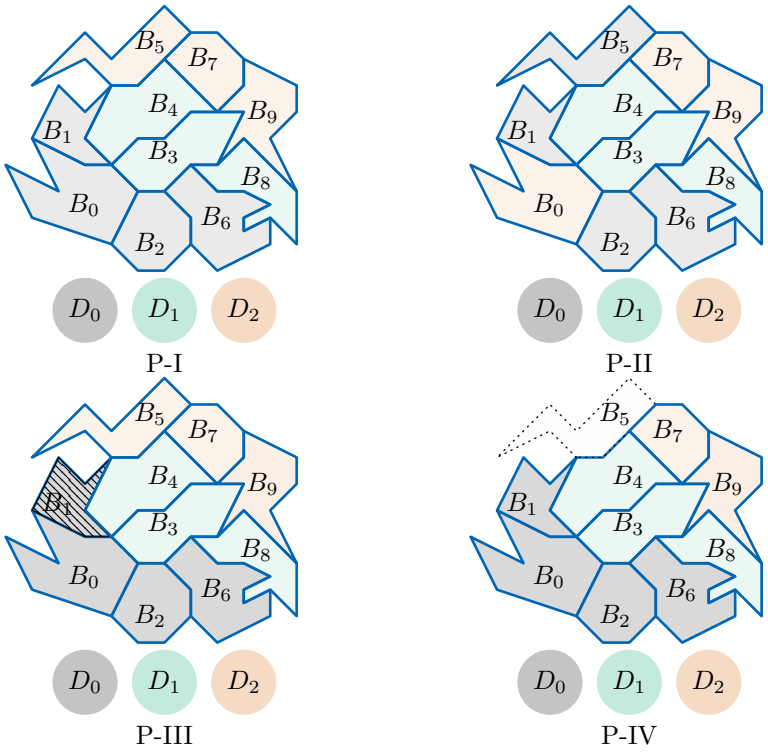

Figure 6.3

Blocks $B_0, B_1, ..B_9$ are used to construct 3 districts gray (D_0), green (D_1), and red (D_2). Four arbitrary redistricting plans are constructed: P-I, P-II, P-III and P-III.

associated with a state redistricting structure. In what follows, we discuss the BREP protocol based on the following assumptions regarding input data:

1. No block contains a hole polygon. In other words, polygons are bordered externally to each other without containing one into the other.
2. Blocks appropriated for redistricting form a contiguous region called (say) state.

A BREP process employs three major sub-processes:

1. State Construction Sub-process Psc,
2. District Plan Construction Sub-process Ppc, and
3. District Plan Evaluation Sub-process Ppe

Given a 4-tuple $(R, \mathcal{D}, A, C)$ defined by a state authority U_p, sub-process Psc specifically operates in the following order to-

1. Initialize blockchain ledger for a state construction sub-process,
2. Initialize OMTs $\mathcal{T}_R$ and $\mathcal{T}_\mathcal{D}$ representing region R, and district set $\mathcal{D}$.
3. Load R into $\mathcal{T}_R$; load $\mathcal{D}$ into $\mathcal{T}_\mathcal{D}$; load A into $\mathcal{T}_R$ (it will be clear later);
4. Trigger sub-process Ppe.

The blockchain network is now ready to receive districting solutions from any non-state authority identified by a public key U_g. Each independent U_g operate to constructs an independent mapping $\mathcal{G}(R, \mathcal{D})$ and the corresponding district plan metric structure (DPMS) $(\Omega, G, \mathbf{M})_{\mathbf{Ug}}$, which is self-claimed 'goodness' value of the plan $\mathcal{G}$. Under sub-process Ppc,

5. U_g broadcast the DPMS over the blockchain network (BN).
6. The BN reaches the consensus to select the best performing plan.

Given a selected best performing plan $\mathcal{G}(R, \mathcal{D})$ purposed by a non-state authority U_g, the sub-process Ppe operates to

7. Validate $\mathcal{G}(R, \mathcal{D})$ against the SRS $(R, \mathcal{D}, A, C)$,
8. Recompute DPMS to validate claimed DPMS $(\Omega, G, \mathbf{M})_{\mathbf{Ug}}$. The computed matrices are again committed to a blockchain ledger.

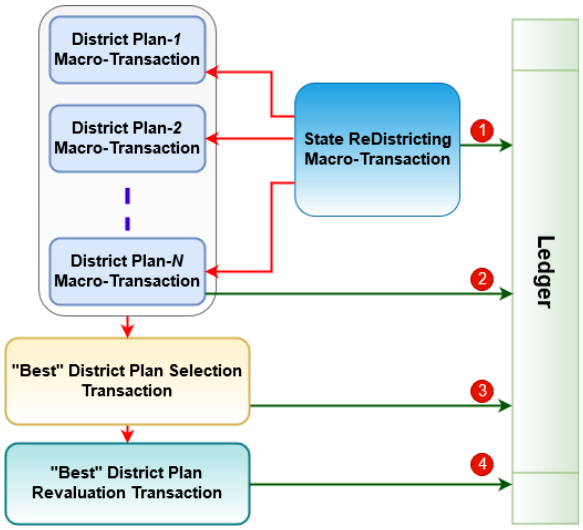

Figure 6.4

BREP as a set of Macro-Transactions

We execute each of these sub-processes as the processes' *macro-transactions* in a BN. Firstly (see Figure 6.4), the macro-transaction for Psc is executed as State Redistricting Macro-Transactions to reach a consensus on the correctness of a Psc. It is followed by the execution of District Plan Macro-Transactions for Ppc to reach a consensus on the correctness of each Ppc; Finally, the Macro-Transactions for Ppe is executed i) to select the "Best" district plan and ii) to validate the selected district plan to finally commit it into a blockchain ledger. Each of the macro-transaction is broken

down into constituent low-complex operations called micro (μ)-transactions. In the later section, we identify specific *μ-transactions* in each of the maco-transactions. In the following section, we discuss data structures that are useful to execute (μ)-transactions in a BN.

6.4 BREP: Data and Transactions

Major actors in a BREP protocol are two types of authorities: i) State authority, U_p, and ii) Non-state authority, U_g. We use a public key U_p^u to identify a state authority U_p. The public key U_p^u is bound with a private key U_p^r. Any non-state authority U_g is identified (as 'g' in general) by a public key U_g^u, is bound with a private key U_g^r. A transaction T_t digitally signed by an authority U_p is denoted as $(T)_{U_p}$; and that signed by U_g is denoted by $(T)_{Up}$. Data required for a state redistricting are defined as State Redistricting Structure (SRS), and data submitted as redistricting plan are described as State Districting Plan (SDP). On the bedrock of BREP are multiple OMTs that facilitate to track the integrity of an SRS, an SDP, and other useful data for a state redistricting project.

6.4.1 State Redistricting Structure (SRS)

A state authority U_p defines a state redistricting structure (SRS). It is denoted as $(\Omega)_{Up}$, which is a 5-tuple $\Omega(R, D, C, A)_{U_p}$, with usual definitions of R, D, C, and A. Corresponding to a 5-tuple Ω_{U_p} SRS apportioned by authority U_p, a blockchain network initiates, maintains, and updates the following OMTs.

1. Block OMT: It is an OMT $\mathcal{T}_{B_i}$ that collects border points for each block $B_i \in R$; with dynamic root σ_{B_i},

2. Region OMT: It is an OMT $\mathcal{T}_R$ that collects 1D-OMT root $\sigma_{B_i} \in R$; with dynamic root σ_R,
3. District OMT: It is an OMT $\mathcal{T}_D$ that collects every district $D_i \in D$; with dynamic root σ_D,
4. Constraint OMT: It is an OMT $\mathcal{T}_C$ that collects all measurable constraints $c_i \in C$; with a dynamic root σ_C.

6.4.1.1 Block OMT $\mathcal{T}_B$

Given a polygon with a sequence of n unique points $p_0, p_1, ..p_n, p_0$ in CCW, each point is added in that order to a Block OMT. Each leaf of the OMT represents a segment that connects two consecutive points p and p'. Hence, the structure of a leaf in the tree is $(p, p', v = 0)$. The index p is defined as $p = n \parallel (x, y)$, where n is monotonically increasing positional index of the point $p(x, y)$ in the input polygon. Given N census blocks $B_0, B_1, ..B_N$, a set of N Block OMTs $\mathcal{T}_{B_0}, \mathcal{T}_{B_1}..\mathcal{T}_{B_N}$ with their respective roots $\sigma_{B_0}, \sigma_{B_1}..\sigma_{B_N}$ are constructed. The usefulness of the tree lies in generating proofs if two points(segments) are adjacent to each other in a given polygon. The root of the tree is useful in asserting the integrity of the geometric polygonal features under consideration. From now on, we use $(p, p', v) \in B$ to convey that a segment connecting points $p(x, y)$ and $p'(x', y')$ exists in the block tree $\mathcal{T}_B$ with root value σ_B, the point associated with leaf with index p is (x, y), a consecutive point that follows it is $p'(x', y')$. During construction of a block OMT for a block B_i, its area a_i, perimeter, p'_i, and centroid (c_x, c_y) is computed by 6.1, 6.2 and 6.3, respectively.

6.4.1.2 Region OMT $\mathcal{T}_R$

The leaves of a region OMT together represent a geographic region selected for redistricting purposes. Each leaf in a region OMT captures the state of a census block in the region. The leaf corresponding to a block B_i in a region OMT is of the form:

$$(k_i, k_n, v_i),$$

where, k_i is the block key, k_n is key to the next block in the OMT, and v is an n-tuple vector. Specifically, the vector captures essential attributes of the block such as area a, perimeter l, centroid (c_x, c_y), polygon-type $t = \{0, 1\}$ (bit "1" for simple and bit "0" for self-intersecting), population p, and a block OMT root σ_{B_i} for the block. It is represented as:

$$v_i = \sigma_B \parallel p \parallel t \parallel a \parallel l \parallel c_x \parallel c_y \tag{6.12}$$

Initially, only non-geometric attribute like population count is initialized for each block in a region OMT. Thus, v is initialized as:

$$v_i = 0 \parallel p \parallel 0 \parallel 0 \parallel 0 \parallel 0 \parallel 0$$

From now on, we use shorthands $B_i.\sigma_B$ to access the root of Block OMT constructed for the block B_i. Similarly, we use $B_i.\times$ to access properties such as p, t, a, l, c_x, and c_y of the block B_i from a leaf in the region OMT.

6.4.1.3 District OMT $\mathcal{T}_D$

District OMT is constructed to maintain the integrity of district identifiers allocated for redistricting the census blocks. Given a set of q district identifiers $\{\mathcal{D}_0, \mathcal{D}_1, ..\mathcal{D}_q\}$, an OMT has q leaves of the form $(d_i, d_n, 0)$, where d_i is the ith district identifier, and d_n is the district identifier of the next district.

6.4.1.4 Constraint OMT $\mathcal{T}_D$

Similar to a district OMT, a constraint OMT maintains the integrity of the constraint variables to be computed for each redistricting plan. Each leaf of this OMT is a structure (c_i, c_n, z), where c_i is the index of ith measurable constraints, c_n is the index of next measurable constraint, and z is the function definition of the constraint. From now on, we use notations $\Omega.R$, $\Omega.\mathcal{D}$, $\Omega.\mathcal{C}$, $\Omega.A$, and so on refer/access parameters defined for an SRS $(\Omega)_{U_p}$.

Once an authority U_r completes defining a redistricting problem Ω_{U_r}, it digitally signs the problem's data object. The signed problem object is broadcast over a block-chain network as a transaction object. The incentivized users in a blockchain network work on to validate such transactions before finally appending the resultant state changes into ledger called redistricting problem chain, called as an $\Omega-$chain, C_Ω.

6.4.2 State Districting Plan (SDP)

A non-state authority U_g constructs a state (re)districting plan (SDP), which is a mapping $\mathcal{G}(R, D)$. We can propose any number of SDPs can be for an SRS. Formally, an SDP is structured as a key-value pairs:

$$\mathcal{G}_{U_g} = \{(b_0, d_i), ..(b_i, d_i)\}, \tag{6.13}$$

where b_i is a block identity of the ith block in a $\Omega.R$, and $\mathcal{D}_i$ is district identity of the ith district in $\Omega.\mathcal{D}$. A districting plan is well-formed if and only if (i) it contains all blocks, (ii) each block is assigned to a distinct district, and (iii) all districts form a contiguous geographic region. For the state redistricting structure mentioned in example-1; few possible SDPs are P-I, P-II, P-III and P-IV as shown in Figure 6.3. Meanwhile, it is useful to notice that only plan P-I satisfies all conditions (i), (ii) and (iii). The districting plan P-II violates the condition (iii) as district $\mathcal{D}_2$ contains a fragmented block B_0. The plan P-III violates the condition (ii) as it assigns block B_1 to two different districts $\mathcal{D}_0$ and $\mathcal{D}_1$. Similarly, the plan P-IV violates the condition (i) because it misses to assign the block B_5 to any district.

An U_g initiates and maintains the following OMTs to maintain the integrity and validity of an SDP $\mathcal{G}$ that is broadcast over a BN.

1. OMT $\mathcal{T}_G$ collects records of $\mathcal{G}_{U_g}$, whose static root σ_G is signed by the non-state authority U_g.
2. Plan OMT: It is an OMT $\mathcal{T}_P$ that collects geometric properties for all districts constructed under a district plan $\mathcal{G}$, with a dynamic root σ_P.

6.4.2.1 Plan OMT $\mathcal{T}_P$

A leaf in a plan OMT $\mathcal{T}_P$ captures the state of a redistricting plan $\mathcal{G}$. It has a leaf of the form:

$$(d_i, d_n, w)$$

, where $d_i \in R.D$ is the district identity of the ith district, and d_n is the next district following this leaf. The value w is defined to include geometric properties such as a simplicity t, an area (a), a perimeter l, and centroid (c_x, c_y). Other properties such as an aggregate of the areas, an aggregate of populations, and the new centroid of the constituent blocks are tracked by variables a', p', and (c'_x, c'_y), respectively. To denote that if a set of blocks constructs a contiguous district, we also include a fragmentation property $f = \{0, 1\}$ (bit "1" for contiguous and bit "0" for fragmented or non-contiguous). Thus the value w_i takes the following form:

$$w_i = f \parallel t \parallel a \parallel l | c_x \parallel c_y \parallel p' \parallel a' \parallel c'_x \parallel c'_y \tag{6.14}$$

From now on, we use $\mathcal{G}.\sigma_G$ to access the static root of the OMT $\mathcal{T}_G$, $\mathcal{G}.(B_i)$ to access an image of the block B_i under the plan $\mathcal{G}$ (or district that is mapped from block B_i), $\mathcal{G}.(\mathcal{D}_i)$ to access all the blocks mapped to a district $\mathcal{D}_i$ in the plan $\mathcal{G}$. We use also use usual notation such as $\mathcal{D}_i$.x to access properties such as t, a, l, a', p', c'_x and c'_y associated with a district $\mathcal{D}_i$.

6.4.3 District Plan Metric Structure (DPMS)

Corresponding to an SRS $(\Omega)_{Up}$ that a state authority U_p constructs, U_p also computes a set of measurable constraints functions defined by $\Omega.C$ to produce a district plan metric structure (DPMS). A DPMS for an SDP $\mathcal{G}$ corresponding to an SRS Ω is a 3-tuple

$$\mathcal{M} = (\Omega, \mathcal{G}, \mathbf{M} = (m_{ij}) \in \mathbb{R}^{q \times k}) \tag{6.15}$$

where expression $\mathbf{M} = (m_{ij}) \in \mathbb{R}^{q \times k}$ is evaluated k measurable constraints $\Omega.\mathcal{C} = \{c_0, .., c_k\}$ for q districts $\{\mathcal{D}_0, ..., \mathcal{D}_q\}$ in $T^{\Omega}.\mathcal{D}$. $\mathbf{M}$ is a matrix with $q \times k$ values of k constraint functions for q district. Corresponding to a district plan, U_g also computes a scalar value $\mathbf{F}(\mathbf{M})$. $\mathbf{F}(.)$ is a function that signifies the compactness and contiguity of a district plan. This scalar value is used to order any given number of submitted SDPs for an SRS, and determine the "best" SDP.

A metric OMT $\mathcal{T}_{\mathcal{M}}$ with root value $\sigma_{\mathcal{M}}$ is used to commit the integrity of computed metrics for an SDP. $\mathcal{T}_{\mathcal{M}}$ collects values m_{ij} measured for constraints c_i and district $\mathcal{D}_j$. Specifically, the OMT collects 3-tuple entries of the form:

$$(d_i, d_n, m_{00} \parallel m_{01} \parallel m_{02}..m_{qk})$$

as its leaves. From now on, we use $\mathcal{G}.\mathcal{M}$ to refer DPMS for an SDP $\mathcal{G}$; conversely $\mathcal{M}.\mathcal{G}$ to get an SDP $\mathcal{G}$ corresponding to $\mathcal{M}$.

6.5 BREP: Macro-Transactions

Having discussed the data structures appropriated for BREP, we discuss the BREP sub-processes as macro-transactions. Macro transactions are broken into smaller, lower-complex atomic operations called micro-transactions. Each micro-transactions can be safely verified its correctness in constant time, with minimal verification objects. Due to the need that a large number of micro-transactions must be executed for each sub-process, it is useful to group logically related micro-transactions together to form what is called as macro-transaction. Each macro-transaction is again be verified in constant time using constant size verification objects. Once a macro-transaction is executed, a blockchain network must reach-global consensus on its output. In turn, an agreed-upon macro-transaction output is stored into a blockchain, which is discussed in later sections.

6.5.1 Macro-Transactions for Sub-Process Psc

A state authority U_p feeds an SRS over a blockchain network as a macro-transaction T^{Ω} defined as:

$$T_i^{\Omega} = [t_i, \delta_t, \Omega]_{U_p} \tag{6.16}$$

which is digitally signed by U_p, at time t_i. It is a public announcement for districting plans from any individuals U_gs. Incentivized users in a blockchain network compete to verify if T_i^{Ω} is well-formed. A transaction T_i is well-formed subject to the following conditions:

1. U_p's signature is valid for the transaction T_i^{Ω}, and

2. Transaction expiry time $(t_i + \delta_t)$ is not reached, and

3. Every census block $B_i \in T^{\Omega}.R$ is a simple polygon. In a simple polygon, no non-adjacent segments intersect with each other. As shown in Figure 6.6, polygon $AB..HA$ is a simple polygon whose segments only intersect at their end-points. Other polygon $IJ..PI$ is not a simple polygon because two non-adjacent segments $PO/(MN)$ and LM intersect. However, a trustworthy mechanism to detect a simple polygon is not a trivial process. In section 6.6, we discuss trustworthy methods for detecting a simple polygon in a blockchain network.

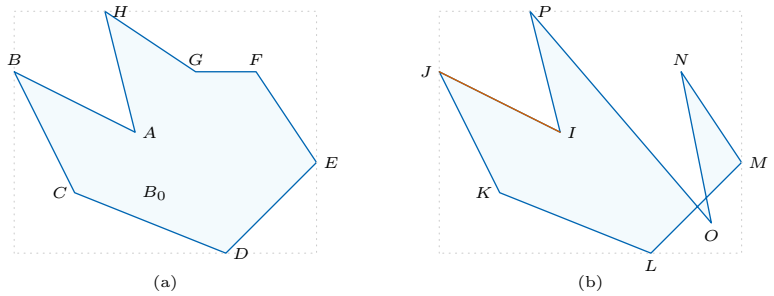

Figure 6.5

(a): A simple polygon $ABC..HGA$; (b): A self-intersecting polygon $IJK..PI$.

6.5.1.1 Micro-Transactions

In section 6.6, we discuss AddPoint(), AddEvent(), DelEvent(), AddActSeg() μ-transactions to execute trustworthy methods to construct and determine if a given block is a simple polygon. In parallel, μ-transactions such as IncrArea(), IncrPeri(), and IncrCentr() are executed to compute area, perimeter, and a centroid of a polygonal block.

6.5.2 Macro-Transactions for Sub-Process *Ppc*

A non-state authority U_g submits DPMS $\mathbf{M}_{Ug}$ corresponding to a DPS $\mathcal{G}$ in the form of macro-transaction

$$T_j^{\mathcal{M}} = [t_j, \mathbf{M}_{\mathcal{G}}, T_i^{\Omega}]_{U_g}; \tag{6.17}$$

To mark the receipt of a DPMS for evaluation, the blockchain network enters the record for this transaction in C_{Σ} ledger. All submitted $\mathbf{M}_{\mathcal{G}}$ are ordered based upon their $\mathbf{F}(.)$ values to reach a consensus on the "best" performing DPMS $\hat{\mathbf{M}}_{\mathcal{G}}$.

6.5.3 Macro-Transactions for Sub-Process *Ppe*

Corresponding to the "best" performing DPMS $\hat{\mathcal{M}}_{\mathcal{G}}$, BN request back $U_{\hat{g}}$ the "best" performing SDP, $\hat{\mathcal{G}} = \hat{\mathcal{M}}_{\mathcal{G}}.\mathcal{G}$. $U_{\hat{g}}$ broadcasts $\hat{\mathcal{G}}$ out of the chain by issuing a macro-transaction $T^{\mathcal{G}}$ defined as:

$$T_j^{\mathcal{G}} = [t_j, \hat{\mathcal{G}}, T_i^{\Omega}]_{U_{\hat{g}}}; \tag{6.18}$$

signed by $U_{\hat{g}}$ at time t_j, corresponding to an SDP transaction T_i^{Ω}. A district plan transaction $T_j^{\mathcal{G}}$ is well-formed if the following conditions are satisfied.

Precondition-1: U_g's signature is valid for the transaction $T_j^{\mathcal{G}}$.

Precondition-2: This transaction must enter network before expiring time set for the problem transaction which implies $T_j.t_j \leq (T_j^{\Omega}.t_i + T_j^{\Omega}.\delta_t)$

Precondition-3: Every block in a region R must be mapped to one of the districts in $\mathcal{D}_i \in T_i^{\Omega}.(\mathcal{D})$. As shown in Figure 6.3, SDP P-I satisfies this condition by assigning every block to one of the gray, green, or red districts. An SDP P-IV misses mapping block B_5 to a district, which makes it an illegitimate district plan.

Precondition-4: A block $B_i \in (T_j^{\mathcal{G}}.\mathcal{G})$ must be assigned to a unique district $d_i \in (T_j^{\Omega} \rightarrow \mathcal{D}_k)$. This property ensures that the mapping assigns a block to a distinct district. As shown in Figure 6.3, SDP P-III is invalid because of the block B_1 is assigned to both gray and green districts.

Precondition-5: Blocks assigned to each district $\mathcal{D}_i$ must construct a geographically contiguous district. In other words, no district is geographically separated into two or more regions. As shown in Figure 6.3, SDP P-II is invalid because of the red district, $\mathcal{D}_2$, is separated into two regions B_0 and $\{B_7, B_9\}$.

6.5.3.1 Micro-Transactions

These validation rules applied by reconstructing a plan OMT $\mathcal{T}_P$ for a $\hat{\mathcal{G}}$. Reconstruction process consists of a series of μ-transactions. We define each of the transactions in relation to a redistricting plan $T_j^{\mathcal{G}}.\mathcal{G}$ where a district $\mathcal{D}_i$ is mapped to a set of n' blocks $\mathcal{D}_i = \{B_0^i, B_1^i, ..B_{n'}^i\} \subset R$.

1. IncrArea($\mathcal{D}_i, B_i$): Given an initialized variable, $a' \in w \in \mathcal{T}_P$ tracking the sum of the area of all blocks $B_i \in \mathcal{D}_i$, it reads region OMT $\mathcal{T}_R$ leaf (B_i, v_i) (the key is B_i) to access area value $B_i.a$, and increments the current value of a' by $B_i.a$.

2. IncrCentr($\mathcal{D}_i, B_i$) Given an initialized variable, $c'_x \in w \in \mathcal{T}_P$ tracking the sum of centroids of all blocks $B_i \in \mathcal{D}_i$, it reads $\mathcal{T}_R$ leaf (B_i, v_i) (the key is B_i) to access centroid value $(B_i.c_x, B_i.c_y)$, and increments the current value of c'_x, c'_y by $(B_i.c_x, B_i.c_y)$.

3. IncrPop($\mathcal{D}_i, B_i$) Given an initialized variable, $p' \in w \in \mathcal{T}_P$ tracking the sum of the population of all blocks $B_i \in \mathcal{D}_i$, it reads $\mathcal{T}_R$ leaf $B_i, v_i)$ (the key is B_i) to access population value $B_i.p$, and increments the current value of p' by $B_i.p$.

4. AddToUnion($\mathcal{D}_i, B_i$) Given a block B_i and union $U_{\mathcal{D}_i}$, this transaction computes the geometric union of B_i and $U_{\mathcal{D}_i}$.

5. CalcUArea($\mathcal{D}_i$): Given an initialized variable, $a \in w \in \mathcal{T}_P$ tracking the area a district $\mathcal{D}_i$, computes the area of the district.

6. CalcUPeri($\mathcal{D}_i$): Given an initialized variable $l \in w \in \mathcal{T}_P$ tracking the perimeter of a district $\mathcal{D}_i$, computes the perimeter of the district.

7. CalcUCentr($\mathcal{D}_i$): Given an initialized variable $(c_x, c_y) \in w \in \mathcal{T}_P$ tracking the centroid district $\mathcal{D}_i$, computes the centroid of the district.

The completion of the above μ-transactions enables to evaluate the rules in **Precondition-3**, **Precondition-4**, and **Precondition-5** mentioned in the earlier section. They are validated as:

1. The aggregate of the areas of constituting blocks for each district in the plan $\mathcal{G}$ is equal to the area of the geometric union of the blocks. In other words,

$$\forall \mathcal{D}_i \in \mathcal{G}, \mathcal{D}_i.a = \mathcal{D}_i.a'; a, a' \in w_i \in \mathcal{T}_P \tag{6.19}$$

   The satisfaction of the above expression ensures that no duplicate block is assigned to a district $\mathcal{D}_i \in R.\mathcal{D}$.

2. The sum of areas of all districts is equal to the sum of areas of all blocks in the region R. In other words,

$$\sum_{i=0, \mathcal{D}_i \in \mathcal{G}}^{q} \mathcal{D}_i.a = \sum_{i=0, B_i \in R}^{N} B_i.a \tag{6.20}$$

   where $\mathcal{D}_i.a \in w \in \mathcal{T}_P, B_i.a \in \mathcal{T}_R$.

   The satisfaction of the above expression ensures that all blocks in the given district have been assigned to at least one district $\mathcal{D}_i \in R.D$.

3. The sum of the areas of all the districts constructed is equal to the area of the union of the districts (i.e., the area of the constructed state). In other words,

$$A_s = \sum_{i=0}^{q} \mathcal{D}_i.a \tag{6.21}$$

Once the macro-transaction $T^{\mathcal{G}}$ is successfully executed for the district plan $\hat{\mathcal{G}}_{U_g}$, a new macro-transaction T^M is triggered to access the correctness of $\hat{\mathbf{M}}_{\mathcal{G}}$. T^M is defined as:

$$\hat{T}^M = [\Omega, \hat{\mathcal{G}}]_{U_g} \tag{6.22}$$

Micro-transactions for $\hat{T}^M$ is a series of $m_{ij} = \textsc{CalcConstF}(\mathcal{D}j, c_i)$ for all $0 \leq i < k$ and $0 < j < q$, invoked to compute each constraint function $c_i \in C$ for each

district $\mathcal{D}_j \in D$. m_{ij} values are stored as leaves of a metric plan OMT $\hat{\mathcal{T}}^{\mathcal{M}}$ with the root value of $\hat{\sigma}_{\mathcal{M}}$. If $\hat{\sigma}_{\mathcal{M}}$ is equal to the root value $\mathcal{T}^{\mathcal{M}}.\sigma_{\mathcal{M}}$–initially claimed by non-state authority U_g, then the district plan $\hat{\mathcal{G}}$ is finally recorded as the "best" agreed by the blockchain network.

6.6 Trustworthy Verification of a Simple Polygon: Using Modified Shamos Hoey Algorithm

In a simple polygon, no non-adjacent segments intersect with each other. As shown in Figure 6.6, polygon $AB..HA$ is a simple polygon whose segments only intersect at their end-points. Other polygon $IJ..PI$ is not a simple polygon because two non-adjacent segments $PO/(MN)$ and LM intersect. However, a trustworthy mechanism to detect a simple polygon is not a trivial process. In this section, we discuss a trustworthy process for detecting a simple polygon in a blockchain network.

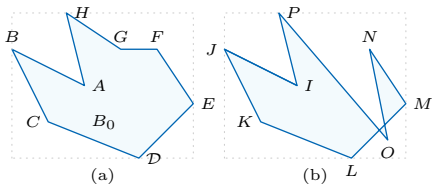

Figure 6.6

(a): A simple polygon $ABC..HGA$; (b): A self-intersecting polygon $IJK..PI$.

The incentivized users in a blockchain reach a consensus on a single commitment until no intersection is detected while processing a given polygon. The protocol makes

use of multiple OMTs to monitor the integrity of input polygon, and integrity of the process under execution. Specifically, we leverage three OMTs to execute Shamos-Hoey algorithm to enable the blockchain network to reach a consensus regarding if a polygon is simple.

- **Block OMT** $\mathcal{T}_B$ with a dynamic root value of σ_B, captures segments (points) on the boundary of an input polygon B. Its construction process is the same as discussed in section 6.4.1.1.
- **Event OMT** $\mathcal{T}_E$ with a dynamic root value of σ_E, to capture the integrity of the process sequence.
- **Active Segment OMT** $\mathcal{T}_A$ with a dynamic root value of σ_A, to capture the integrity of a state of processing input segments.

In the following sections, we discuss the construction and the utility of these OMTs.

6.6.0.1 Event OMT $\mathcal{T}_E$

Construction of an Event OMT begins once construction of a Block OMT completes. Each leaf of a Block OMT $\mathcal{T}_B$ is processed to insert two event leaves into an Event OMT $\mathcal{T}_E$. Given a leaf $(p, p', v) \in \mathcal{T}_B$ representing a segment connecting two points (x, y) and (x', y'), two event leaves inserted in an event tree are:

1. $(x, \alpha_n, x \parallel x' \parallel y \parallel y' \parallel S)$ and $(x', \alpha'_n, v = x \parallel x' \parallel y \parallel y' \parallel E)$ if $x < x'$

2. $(x, \alpha_n, x \parallel x' \parallel y \parallel y' \parallel E)$ and $(x', \alpha'_n =, v = x \parallel x' \parallel y \parallel y' \parallel S)$ if $x > x'$

where, encoding S denotes a 'start' event leaf, and E denotes an 'end' event leaf; α_n and α'_n denote indices of succeeding leaf. Given N Block OMTs $\mathcal{T}_{B_0}, \mathcal{T}_{B_1}..\mathcal{T}_{B_N}$, it results in N Event OMTs $\mathcal{T}_{E_0}, \mathcal{T}_{E_1}..\mathcal{T}_{E_N}$ with respective roots.

From now on, we use $(\alpha, \alpha_n, v) \in \mathcal{T}_E$ to convey that the leaf exists in the event tree $\mathcal{T}_E$ with root value σ_E, α."LeafType" to access encoding S or E associated with the event leaf with index α; $\alpha.(x, y)$ and $\alpha.(x', y')$ to access segment endpoints (x, y) and (x', y') encoded in v.

6.6.0.2 Active Segment OMT $\mathcal{T}_A$

An Active Segment OMT is due for construction immediately after the construction of the Event Tree is completed. The leaves in $\mathcal{T}_E$ are processed in order (from the lowest key leaf to the highest key leaf) to insert into an active segment OMT $\mathcal{T}_A$. Each leaf in the tree is of form the (γ, γ_n, v). Given a leaf $(x, \alpha_n, x \parallel y \parallel x' \parallel y' \parallel \{S, E\}) \in \mathcal{T}_E$, two major operations are performed for inserting/deleting a new leaf into this tree.

- if α.EventType is S, then it then a leaf of the form $(y, \gamma_n, \mu = x \parallel y \parallel x' \parallel y')$ is inserted in $\mathcal{T}_A$. Delete $(x, \alpha_n, x \parallel y \parallel x' \parallel y' \parallel S)$ from $\mathcal{T}_E$.
- if α.EventType is 'E', then existing leaf γ is deleted/removed from $\mathcal{T}_A$. Delete leaf $(x, \alpha_n, x \parallel y \parallel x' \parallel y' \parallel S)$ from $\mathcal{T}_E$.

The construction of the tree dictates sequential operations of vertices in a polygon from left to right. It ensures that no vertices(segments) are excluded from considering intersections. From now on, , we use $(\gamma, \gamma_n, v) \in \mathcal{T}_A$ to convey that the leaf exists in the active event tree $\mathcal{T}_A$ with dynamic root σ_A. Also, we use $\gamma.(x, y)$ and $\gamma.(x', y')$ to access segment connecting endpoints (x, y) and (x', y') encoded in μ; γ

6.6.1 Blockchain μ-Transactions

Operations for the construction of three OMTs explained in the previous section are executed as a series of verifiable μ-transactions in a BN. Initially, a blockchain network bootstraps by initializing the state of each of the OMTs to zeros such as

$$(\sigma_B \leftarrow 0, \sigma_E \leftarrow 0, \sigma_A \leftarrow 0, \xi_B \leftarrow 0)$$

where bit variable $\xi_B = 0, 1$ is "1" if a block is simple. Well-defined operations are executed in a blockchain, attempting to set this variable to "1" to suggest that a BN successfully reached consensus about the polygon being simple.

We define each of the transaction related to a polygonal block B_i with n points $B_i = \{p_0, p_1, ..p_n, p_0\}$ in CCW; each point p_i is a coordinate (x, y).

1. AddPoint(i, x, y): Inserts ith point (x, y) in Block OMT $\mathcal{T}_B$. A process invokes this transaction for all points in the block. The first call of this transaction initializes a Block OMT with initial root value 0. Subsequent calls progress the root value until the last invocation is made, which marks the end of the insertion process.

2. IncrArea(x_i, y_i, x_j, y_j): Given an initialized variable, a tracking an area of an input block B_i, this transaction increments the current value of a by $(x_i + x_{i+1})(y_{i+1} - y_i)$.

3. IncrPeri(x_i, y_i, x_j, y_j): Given an initialized variable p tracking perimeter of an input block B_i, this transaction increments the current value of p by

$$\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

4. IncrCentr(x_i, y_i, x_j, y_j): Given an initialized variables c_x and c_y tracking a centroid (c_x, c_y) of an input block B_i, this transaction increments the current value of c_x by $(x_i + x_{i+1})(x_i y_{i+1} + x_{i+1} y_i)$ and the current value of c_y by $(y_i + y_{i+1})(x_i y_{i+1} + x_{i+1} y_i)$.

5. AddEvent(p, p', v): Adds two events associated with the points p and p' subject to the following rules:

- $(p, p', v) \in \mathcal{T}_E$
- If an event with key $p.x \in \mathcal{T}_E$, single update value v to v' of an event leaf$p.x, \alpha_n, v$ as: if $p.x < p'.x$, $v' = v \parallel p.x \parallel p.y \parallel p'.x \parallel p'.y \parallel S$, otherwise, $v' = v \parallel p.x \parallel p.y \parallel p'.x \parallel p'.y \parallel E$. Otherwise, insert new leaf with key $p.x$.

6. DelEvent(p): Deletes an event with key $p.x$ if exists in event OMT $\mathcal{T}_E$.

7. AddActSeg($l_e : (\alpha, \alpha_n, v)$): Adds a segment s_i connecting two points $\alpha.(x, y)$ and $\alpha.(x', y')$ to an Active Event OMT $\mathcal{T}_A$ subject to the following conditions:

   - α.EventType is "S."
   - Event with key $y \notin \mathcal{T}_A$, else abort the process with the state $(\sigma_A, \sigma_E, \sigma_A, 1)$.
   - IAbv($l_e : (\alpha, \alpha_n, v), l_a : (\gamma, \gamma_n, v')$): Active segment s' $l_a : (\gamma, \gamma_n, v') \in \mathcal{T}_A$ is just above the current segment s_i; and s and s' do not intersect. If they intersect, then abort the process with the state- $(\sigma_A, \sigma_E, \sigma_A, 1)$.
   - IBel($l_e : (\alpha, \alpha_n, v), l_a : (\gamma, \gamma_n, v')$): Active segment s' $l_a : (\gamma, \gamma_n, v') \in \mathcal{T}_A$ is just below the current segment s_i; and s and s' do not intersect. If they intersect, the process is aborted with the state- $(\sigma'_B, \sigma'_E, \sigma'_A, \xi_B : 0)$.

8. DelActSeg($(\alpha, \alpha_n, v), (\gamma', \gamma'_n, v'), (\gamma'', \gamma''_n, v'')$): Deletes a segment s_i connecting two points $\alpha.(x, y)$ and $\alpha.(x', y')$ from an Active Event OMT $\mathcal{T}_A$ subject to the following conditions:

   - α.EventType is "E".
   - Event with key $y \in \mathcal{T}_A$
   - Active segment s' $l_a : (\gamma', \gamma'_n, v') \in \mathcal{T}_A$ which is just above the current segment s_i; Thus, $\gamma'.y = \alpha.\alpha_n$ is true.
   - Active segment s'' $l_a : (\gamma'', \gamma''_n, v'') \in \mathcal{T}_A$ which is just below the current segment s_i; Thus, $\gamma'' = \alpha$ is true.
   - Segments s' and s' are not adjacent segments in $\mathcal{T}_B$.
   - Segments s' and s' do not intersect. If they intersect then abort the process with the state $(\sigma'_B, \sigma'_E, \sigma'_A, \xi_B : 0)$

   Whenever all leaves in an Event OMT $\mathcal{T}_E$ are processed, and still the blockchain state is still found to be $(\sigma_B : 0, \sigma_E : 0, \sigma_A : 0, \xi_B : 0)$, a consensus is reached on a current roots σ'_B, σ'_E and σ'_A, of OMTs as a new state as:

   $$(\sigma'_B, \sigma'_E, \sigma'_A, \xi_B : 1)$$

   to mark that the block is identified to be a simple polygon.

In parallel to building a Block OMT for a census block, we compute and update several geometric properties such as area, perimeter, and the centroid of the polygonal block. For instance, we compute area, perimeter, and centroid iterating over vertices of a simple polygon.

6.7 Trustworthy Verification of a Simple Polygon: Bounding Box Method

The process starts by adding every point into a Block OMT (as discussed in section 6.4.1.1). Corresponding to polygonal block B_i, a set of non-overlapping bounding boxes (BBs) that covers the polygonal region and can safely enclose all the boundary segments in the block is determined. In other words, each segment in the polygonal block can be mapped to one of the BBs. As a rule, only two segments can be mapped to a BB only if the segments meet at one of the corners of the BB. In case, a segment mapped to a BB intersects (excluding at the corner) with one of the previously mapped segment, a self-intersecting polygon is identified. This operation is performed with the following transactions.

1. $\mathsf{AddPoint}(\mathsf{p}, \mathsf{p}')$ to add every point $p \in B_i$ into a Block OMT, T_{B_i}, with a dynamic root σ_{B_i}. To add all points in B_i, a sequence of

   $$\mathsf{AddPoint}(\mathsf{p_0}, \mathsf{p_0}), \mathsf{AddPoint}(\mathsf{p_1}, \mathsf{p_0})..\mathsf{AddPoint}(\mathsf{p_n}, \mathsf{p_0})$$

   are executed. The root of the OMT is updated accordingly.

2. $\mathsf{InitBB}(\mathsf{BB_i})$ to initialize an OMT T_{BB} with dynamic root σ_{BB}, where B_i is 5-tuple (x_l, y_l, x_h, y_h, v) representing bounding box for a block B_i.

3. $\mathsf{SplitBB}(\mathsf{BB_j} = (\mathsf{x_a}, \mathsf{y_a}, \mathsf{x_b}, \mathsf{y_b}), [\mathsf{x}', \mathsf{y}'])$ to split a BB BB_j along vertical split axis $x = x'$ or horizontal axis $y = y'$. For instance, splitting a BB_j along $x = x'$ produces two BBs (x_a, y_a, x', y_b) and (x', y_a, x_b, y_b). The root of the OMT is updated accordingly.

4. $\mathsf{InterPolate}(\mathsf{pp'}, \mathsf{q})$ inserts a new point q between two points p and p' in block-OMT T_B whenever $\mathsf{SplitBB}(\mathsf{BB_j}, [\mathsf{x'}, \mathsf{y'}])$ intersects existing segment $pp' \in B_i$. The root of the 1-D OMT T_B is updated accordingly.

5. $\mathsf{MapLine}(\mathsf{BB_j} = (\mathsf{x_a}, \mathsf{y_a}, \mathsf{x_b}, \mathsf{y_b}, \mathsf{pp'}))$ to map a segment pp' to a $BB \in T_{BB}$. Mapping a segment to a BB is setting the value of the leaf by the segment identifier. Let $BB_j.v$ be a segment mapped to this BB. The preconditions for this transaction are:

   - pp' is a diagonal of BB_j and $BB_j.v = 0$ (empty BB_j), OR
   - One of p and p' is a corner and other is on the side of BB_j, and $BB_j.v = 0$ or $BB_j.v = 0$ is a diagonal segment.
   - pp' does not intersect between p and p' with a segment previously mapped to BB_j

   If segment pp' intersects previously mapped segment $qq' \in T_B$, the transaction is aborted to identify a self-intersecting polygon.

For example, a μ-transaction $\mathsf{InitBB}(\mathsf{01})$ initializes a bounding box for B_0 in Figure 6.7(a). Then, μ-transaction $\mathsf{SplitBB}(\mathsf{01}, \mathsf{C_x})$ splits BB **01** into two BBs **01.1** and **01.2**, as shown in Figure 6.7(c). This is followed by μ-transaction $\mathsf{InterPolate}(\mathsf{BA}, \mathsf{q_1})$, which inserts a new point q_1 into 1-D OMT $BA \in \mathcal{T}_{B_0}$. Notice that $q1$ is an intersection point between split axis $x = C_x$ and segment $BA \in \mathcal{T}_{B_0}$. This process is continued until the complete set of BBs (as shown in Figure 6.7(d))is built so that all segments/splits in $\mathcal{T}_{B_0}$ can be enclosed. Then segment mapping is triggered. The μ-transaction $\mathsf{MapLine}(\mathsf{02}, \mathsf{BC})$, $\mathsf{MapLine}(\mathsf{02}, \mathsf{Bq_1})$, $\mathsf{MapLine}(\mathsf{04}, \mathsf{CD})$, $\mathsf{MapLine}(\mathsf{10}, \mathsf{DE})$, and so on are used to map the segments to one of BB until all segments in $\mathcal{T}_{B_0}$. Notice that, red shaded BBs contain two segments meeting at a corner, blue shaded BBs contain single diagonal segment, green-shaded BBs contain a segment that is one of the side of the BB and gray shaded are empty BBs.

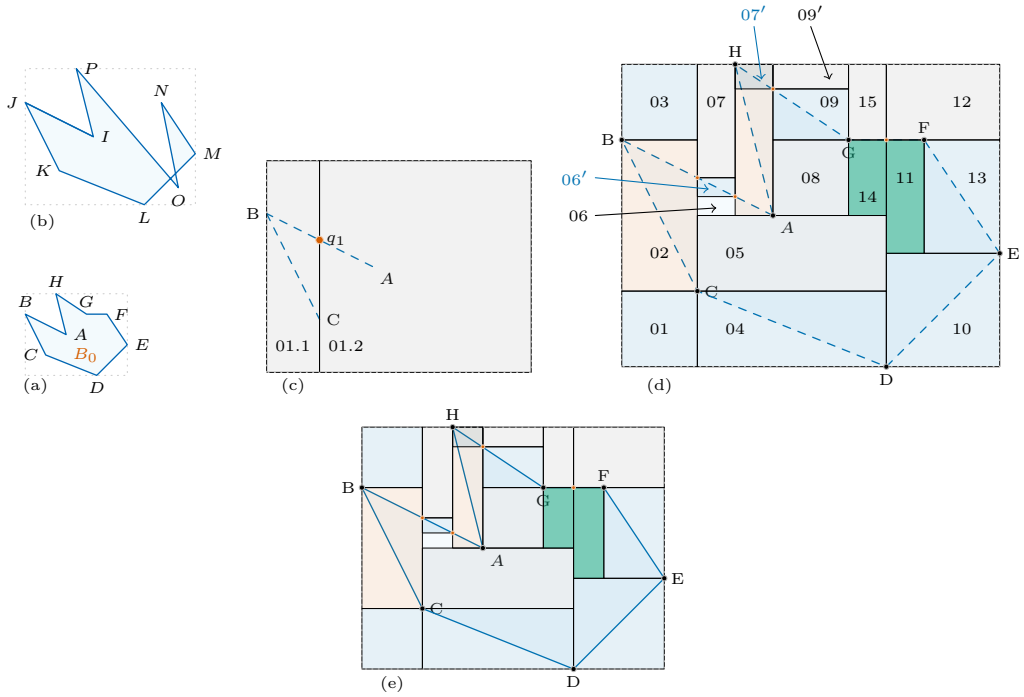

Figure 6.7

(a) A simple polygon $B_0\{A, B, ..H, A\}$ where no non-adjacent segments intersect with each other. (b)Polygon with vertices $\{I, J, ..P, I\}$ is not a simple polygon because two non-adjacent segments $PO/(MN)$ and LM intersect. (c) A set of BBs, T_{BB}=$\{01, 02, 03..07', 09', ..15\}$ built to map segments in polygon B_0. (d) Segments in B_0 is mapped to a BB in T_{BB}

6.8 BREP: Blockchain States

An append-only blockchain ledger is maintained to keep track of BREP processes states. It captures three distinct transaction states: i) commitment for each SRSs, ii) commitment for each DPMS, and ii) a record to the validated and evaluated "best" performing SDP corresponding to an SRS.

Once macro-transaction T_i^{Ω} is completed, BN reaches a consensus on a 5-tuple $(t, \sigma_R, \sigma_D, \sigma_C, \eta, \mu)$ where σ_R is the root value of OMT $\mathcal{T}_{\mathcal{R}}$, $\sigma_{\mathcal{D}}$ is the root value of OMT $\mathcal{T}_{\mathcal{D}}$, and σ_C is a root value representing measurable constraints for a district plan, η is the sequence number, μ is chain accumulation until the previous entry in a chain. The first entry $(0, 0, 0, 0, 0, 0)$ is the genesis entry with the sequence number $\eta = 0$. Hence, C_{Σ} until the ith SRS recorded at timestamp T is a blockchain of:

$$(0,0,0,0,0,0), (t_0, \sigma_R, \sigma_{\mathcal{D}}, \sigma_{\mathcal{C}}, 1, \mu_1)..(t, \sigma_R, \sigma_{\mathcal{D}}, \sigma_{\mathcal{C}}, \eta, \mu_i)$$

where, μ_1 is $h(0, 0, 0, 0, 0, 0)$, and $\mu_{i+1} = h(t_i, \sigma_{R_i}, \sigma_{\mathcal{D}_i}, \sigma_{C_i}, \eta_i, \mu_i)$.

Once macro-transaction $T^{\mathcal{M}}$ successfully executed, it is appended at timestamp t in the same chain as a 5-tuple entry $(t, \sigma_{\mathcal{M}}, \sigma_G, U_g, \sigma_R, \mu)$. Once the BN reaches consensus on the best SDP for an SRS, then it is recorded as a 5-tuple entry

$$(t, 1, \sigma_{\mathcal{M}}, \sigma_G, U_g, \sigma_R, \mu)$$

The incentive for a non-state authority to submit an SDP is that the best performing SDP is rewarded. However, the best performing SDP selected by BN is again vali-

dated, as mentioned in section 6.5.2. If the SDP fails the validation rules, then the authority submitting the SDP loses the stake placed during the submission. These mechanisms encourage any entity to submit a valid SDP for a reward; meantime, it discourages to submit an invalid SDP. The ability of the protocol to store complete SDP out of the chain obviates the storage demand for nodes. Only small transactional records are stored in a ledger by the computing nodes.

6.9 Evaluation Tools and the Methods

Census record of 2010 and spatial data of census-blocks in the "TIGER/Line" data format was accessed from *ftp://ftp2.census.gov/geo/*. The geographic entities such as state, census blocks, and congressional districts are uniquely identified by codes such as Federal Information Processing Series (FIPS) and Geographic Names Information System Identifier (GNIS ID). These codes are adopted by the National Institute of Standards and Technology (NIST) and Census Bureau to identify entities in the geospatial data.

Census block-level spatial data for each US state are publicly available through:

- https://www.census.gov/geographies/mapping-files/2019/dec/rdo/116-congressional-district-bef.html ($SRC1$)
- https://www.maris.state.ms.us/HTM/Data.html
- https://catalog.data.gov/dataset/tiger-line-shapefile-2010-2010-state-alabama-2010-census-block-state-based-shapefile-with-housing;
- ftp://ftp2.census.gov/geo/tiger/TIGER2010BLKPOPHU/tabblock2010_01_pophu.zip

The storage size of processed census blocks spatial file (in cartographic boundary shapefiles (.shp)) ranges from 4MB (District of Colombia) to 760 MB (Texas), with

an average shapefile of 278MB. In total 11.8 GB of census-blocks spatial data was processed. The spatial data extracted from different public sources will be stored in MangoDB and regular comma-separated text files. Spatial data visualization tools such as QGIS, and mapshaper modules are useful. Graphics and visualization are be produced by tools such as tikz library for Latex, or *matplot* library.

We considered a total of 230 congressional districts from 49 US states for evaluation under different metrics such as Iso-Perimetric index, Robharch index, and so on. The spatial boundary and population of the census-blocks in each of the states were pre-processed to transform into a state redistricting structures (SRS). The current congressional districts in the states are used to construct state districting plans (SDP) to the corresponding SRS.

The total number of census blocks were 6643775, with average counts of blocks for a district being 29012; and minimum count of blocks being 1538 in 13th district of New York, and maximum count of blocks being 133769 in single district North Dakota.

6.9.1 GIS Development Tools

All of the software tools and programming libraries identified to be useful for our research are free and open-source. Software modules required for the system development will be written in the Python programming language. Among others, it can leverage supporting program modules like OSGeo, osgr, numpy, scipy, matplotlib, pyshape, geopandas, shapely, scipy.spatial, etc. Test frameworks will be designed and

implemented in python using different test constructs and modules readily available as open and free resources.

6.9.2 Cryptographic Tools and Protocols

Standard cryptographic tools for encryption, decryption, hashing, digital certificate generation, among others, are borrowed from several free and open-source python modules such as crypt, hashlib, and bitcoin.

In this experiment, census blocks of states of the USA were taken to define a state redistricting structure. Current redistricting plans for each of the states were taken as redistricting plans structure (DPS) for each of the redistricting problems. Constraint functions considered for evaluating redistricting plans are IA, IP, HA, RR, MR, DR, etc.

6.10 Related Works, Results, and Conclusions

The time to complete major operations such as computing IPQ, area-moment, population-moment, and Rohbarc indices and block union were measured. The most time-consuming operation was a geometric union of the blocks to construct the district boundary. The operation is critical in constructing a covering boundary of a district given constituent census-blocs. The order of magnitude (base 10) of run time (in the second) for this operation among all districts is at most 2. As the redistricting metrics themselves are insignificant than their secure/trustworthy computation, they are not included as part of the results. BREP offers the following merits:

1. An open evaluation system is a transparent box for public auditing. It lefts redistricting problems to independent untrusted parties, which obviates the

necessity to inspect and evaluate redistricting platforms (hardware and software(s)) from different security perspectives. It only operates in ways to evaluate redistricting plans proposed by untrusted parties.

2. Since it embraces competitive approach, it considers redistricting efforts from independent expert(s) be treated equally under a standard evaluation criteria.
3. As the selection of the optimal redistricting plan is left to the expertise of independent evaluating computing nodes, related disputes between political parties are expected to be nullified.

The use of computer systems to redistricting purpose goes as back as the 1960s. Fully automated, semi-automated, and fully manual have been used in different states for redistricting. While the fully automated system is efficient, it is insecure and cannot be trusted. Traditionally, redistricting has been performed by a committee of experts and political representatives. However, this approach is highly inefficient than an automated redistricting method. Scholars M. Altman and M. McDonald, Karin Mac Donald [3, 4] recommended public participation in producing and verifying optimal redistricting plan. They introduced the concept of an open-access system that provides access to redistricting plans, data, and tools to create such plans to be evaluated publicly. They contended that such a model has the potential to produce thousands of proposals on redistricting and gives way to much transparency and public participation. With an immense potential, such a model immediately introduces problems of evaluating litany of potential district plans. It consumes exhaustive man-hours to evaluate the plans. Hence, it demands automation using a high-performance computer system. However, as argued earlier section, a conventional computing model is prone to produce untrusted and unreliable evaluation metrics for a district plan. This leads to demand for a trustworthy, distributed system for automating the evaluation

of district plans. BREP is a distributed consensus-based computing model with the potential to enhance fairness in a redistricting project. Since redistricting is directly related to public interest, the model is in line with Saeber's public participation geographic information system (PPGIS)[63].

CHAPTER 7

CONCLUSIONS

Information systems are composed of processes for "capturing, transmitting, storing, retrieving, manipulating, and displaying information" [2]. Information is increasingly becoming a valuable commodity. However, the utility of information is ultimately limited by the extent of trust in its validity.

In this dissertation, our focus was on geographic information. A high level of confidence in the integrity of geographic information can enable a wide range of compelling applications, and efficient strategies enhancing citizen trust in governments. One contribution of this dissertation lies in recognizing two important assurances regarding geographic information, viz., i) authoritativeness and ii) unbiasedness.

For information to be *authoritative* it is necessary for the receiver of the information to establish

1. the source of the information, and
2. the chain of delegations through which the source gained the authority to create/propagate such information.

For information to be *unbiased*, it is necessary to establish that no information relevant to the query was suppressed.

Ultimately, all assurances stem from assumptions. More specifically, assumptions are translated to assurances by well designed protocols. In other words, assurances are meaningful only if the assumptions are reasonable, the protocols that leverage assumptions are correct.

Central to the approach adopted in this dissertation is the need to minimize "what we need to trust," in order to validate the correctness of assumptions. More specifically, well founded assumptions like i) standard security properties of cryptographic hash functions (preimage resistance and collision resistance) and ii) the ability to achieve consensus on simple state-changes in a blockchain broadcast network are the only assumptions underlying all protocols and information systems in this research work.

A blockchain network operates by viewing breaking any computable process into a sequence of atomic transactions. Execution of each transaction causes the system to move from mutually agreed, verified, valid old state to a verified, and mutually agreed new system state. The SQDM protocol was developed on top of a blockchain network to guarantee both authoritative and unbiased responses to point location queries. The protocol was then extended to enable Geographic Region Delegation Protocol (GRDP) for delegation of a two-dimensional geographic space (example, state, county, city, land parcel, etc.).

Any geographic feature such as point or line can be delegated in the same way we delegate namespace in a Domain Name System (DNS). SQDM and GRDP can be used to implement automated and federated e-governance services based on geographic

locations or regions. It obviates the need for Trusted Third Party (TTP) between delegator and the delegatee.

As another utility of the SQDM protocol, a novel Blockchain-based Redistricting Protocol (BREP) was designed and evaluated. BREP automates the congressional redistricting task by outputting a highly optimized and trustworthy districting plan from the public sphere.

To summarize, the protocols explored in this dissertation are primarily motivated by the need to eliminate dependence on "trusted third parties" for guaranteeing the integrity of data (example, geographic data assets for the map for geographic delegation purpose) and the process (example, geographic services such as map and point location services) output.

REFERENCES

[1] N. Adhikari, N. Bushra, and M. Ramkumar, "Secure Queryable Dynamic Maps," *The 16th International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government*, Las Vegas, 2017.

[2] S. Alter, "Defining information systems as work systems: Implications for the IS field," *European Journal of Information Systems*, vol. 17, no. 5, 2008, pp. 448–469.

[3] M. Altman, *Districting principles and democratic representation*, doctoral dissertation, 1998.

[4] M. Altman and M. P. McDonald, "The Promise and Perils of Computers in Redistricting," *Duke Journal of Constitutional Law & Public Policy*, vol. 5, no. 1, 2010, pp. 69–159.

[5] S. Arya and D. Mount, "Computational Geometry - Proximity and Location," *Handbook of Data Structures and Applications*, D. P. Mehta and S. Sartaj, eds., 2005 edition, Chapman & Hall/CRC, 2005, pp. 1–63.

[6] S. Banescu and A. Pretschner, "A Tutorial on Software Obfuscation," *Advances in Computers*, 2018.

[7] J. O. N. L. Bentley and T. A. Ottmann, "Algorithms for Reporting and Counting Geometric Intersections," *IEEE Trans. Comput.*, vol. C, no. 9, 1979, pp. 643–647.

[8] E. Bertino, B. Carminati, E. Ferrari, B. Thuraisingham, and A. Gupta, "Selective and Authentic Third-Party Distribution of XML Documents," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 10, 2004, pp. 1263–1278.

[9] P. Bourke, "Calculating the area and centroid of a polygon,", 1988.

[10] B. Braden, "The Surveyor's Area Formula," *The College Mathematics Journal*, vol. 17, no. 4, 1986, pp. 326–337.

[11] N. Bushra, N. Adhikari, and M. Ramkumar, "A TCB minimizing model of computation," *Communications in Computer and Information Science*, vol. 969, no. October, 2019, pp. 455–470.

[12] V. Buterin, "A next-generation smart contract and decentralized application platform,", 2014.

[13] M. Chacos, Brad; Simon, "Meltdown and Spectre FAQ: How the critical CPU flaws affect PCs and Macs,", 2018.

[14] C. P. Chambers, "A Measure of Bizarreness," *Quarterly Journal of Political Science*, vol. 5, no. 1, 2010, pp. 27–44.

[15] B. Chazelle and J. L. Guibas, "Fractional Cascading: II. Applications," *Algorithmica*, vol. 1, no. 4786, 1986, pp. 163–191.

[16] Y.-J. Chiang and R. Tamassia, "Dynamization of the trapezoid method for planar point location," *Proceedings of the Seventh Annual Symposium on Computational Geometry*, New York, USA, 1991, number January, pp. 61–70, ACM.

[17] R. Chiang, Yi; Tamassia, "Dynamic Algorithms in Computational Geometry," *Proceedings of the IEEE*, Providence, RI, USA, 1992, IEEE.

[18] K. C. Clarke, "Map Data Structures," *Analytical and computer cartography*, 1995, pp. 133–156.

[19] V. Cohen-Addad, P. N. Klein, and N. E. Young, "Balanced power diagrams for redistricting," 2017.

[20] T. Consistency, "Chapter 2 . The Consistency and Effectiveness of Mandatory District Compactness Rules," 1998, pp. 989–1012.

[21] I. S. Consortium, *Bind 9 administrator reference manual*, Tech. Rep., Internet Systems Consortium, Inc, 2005.

[22] R. Crocker, *Congressional redistricting: an overview*, Tech. Rep., 2012.

[23] K. David, *GRAPHIC GEMS II Edited by DAVID KIRK*, Academic Press, Inc., Palo Alto, Californrnia, 2012.

[24] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, vol. 17, 2008.

[25] Department of Defense, *Trusted computer system evaluation criteria*, Department of Defense, 1985.

[26] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine, "Authentic Data Publication Over the Internet.," *Journal of Computer Security*, vol. 11, no. 0085961, 2003, pp. 291–314.

[27] E. Dieh, *Ten Laws for Security*, first edition, Springer International Publishing AG, Gewerbestrasse, 2016.

[28] A. Esri and W. Paper, "ESRI Shapefile Technical Description," , no. July, 1998.

[29] R. Franco, Preparata P.; Tamassia, "Fully Dynamic Point Location in a Monotone Subdivision," *SIAM Journal on Computing*, vol. 18, no. 4, 1989, pp. 811–830.

[30] O. Fries, K. Mehlhorn, and S. Naher, "Dynamisation of Geometric Data Structures," *Proceedings of the 1st Annual Symposium on Computational Geometry*, J. O'Rourke, ed. 1985, pp. 168–176, ACM.

[31] S. Gautum, *Cryptanalysis and Design of Symmetric Cryptographic Algorithms*, doctoral dissertation, Katholieke Universiteit Leuven, 2011.

[32] M. T. Goodrich, R. Tamassia, and N. Triandopoulos, "Efficient authenticated data structures for graph connectivity and geometric search problems," *Algorithmica*, vol. 60, no. 3, 2011, pp. 505–552.

[33] G. Hunt, G. Letey, and E. Nightingale, "The Seven Properties of Highly Secure Devices," *Tech. Report Microsoft*, , no. MSR-TR-2017-16, 2017, p. 10.

[34] A. Jacobi, M. L. Jensen, L. Kool, G. Munnichs, and A. Weber, *Security of eGovernment Systems*, Tech. Rep., Science and Technology Options Assessment, Brussels, Belgium, 2013.

[35] L. Knudsen and M. Robshaw, *The Block Cipher Companion*, 3rd edition, Springer, New York, 2011.

[36] S. Krishnaswamy, W. Hardaker, and R. Mundy, "DNSSEC in Practice : Using DNSSEC-Tools to Deploy DNSSEC," *2009 Cybersecurity Applications & Technology Conference for Homeland Security*. 2009, pp. 3–15, IEEE Computer Society.

[37] B. W. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practice," *ACM Tansactions on Computer Systems*, vol. 10, 1992, pp. 265–310.

[38] D. Larochelle and D. Evans, "Statically detecting likely buffer overflow vulnerabilities," *10th USENIX Security Symposium*, 2001.

[39] D. Lee and F. Preparata, "Location of a Point in Planar Subdivision and its Appliccations," *SIAM Journal on Computing*, vol. 6, no. 3, 1977, pp. 5–6.

[40] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A General Model for Authenticated Data Structures," *Algorithmica*, vol. 39, no. 1, 2004, pp. 21–39.

[41] J. M. McCune and A. R. M. K. Perrig, *Reducing the trusted computing base for applications on commodity systems*, doctoral dissertation, Carnegie Mellon University, 2009.

[42] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 2001.

[43] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," *Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, London, UK, 1988, pp. 365—-378, Springer-Verlag.

[44] Mohanty; Somya D., *Ordered Merkel Tree: A Versatile Data-Structure for Security Kernels*, doctoral dissertation, Mississippi State University, 2013.

[45] D. Mount, "CMSC 754 Computational Geometry," *Lecture Notes, University of Maryland*, 2002, pp. 1–122.

[46] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Www.Bitcoin.Org*, 2008, p. 9.

[47] G. T. Nguyen and K. Kim, "A survey about consensus algorithms used in Blockchain," *Journal of Information Processing Systems*, vol. 14, no. 1, 2018, pp. 101–128.

[48] M. H. Overmars and J. van Leeuwen, "Maintenance of Configurations in the Plane," *Journal of Computer and System Sciences*, vol. 23, no. 2, 1981, pp. 166–204.

[49] D. Patten, "The evolution to fileless malware," *Retrieved from*, 2017.

[50] F. Preparata, "A New Approah to Planar Point Location," *SIAM J. Computing*, vol. 10, no. September, 1978, pp. 473–483.

[51] R. Preparata P., Franco; Tamassia, "Dynamic Planar Point Location With Optimal Query Time," *Theoretical Computer Science Science*, vol. 74, no. 1, 1990, pp. 95–114.

[52] M. P.V, "Domain Names- Concept and facilities,", 1987.

[53] R. L. Raja, "Redistricting : Reading Between the Lines," *The Annual Review of Political Science*, 2009.

[54] M. Ramkumar, "Efficient Key Distribution Schemes for Large Scale Mobile Computing Applications," *IACR Cryptology ePrint Archive*, 2008.

[55] M. Ramkumar, "Introduction," *Symmetric Cryptographic Protocols*, 1st edition, Springe, 2014, chapter Introducti, pp. 1–9.

[56] M. Ramkumar, "Cybersecurity: It's All About Assumptions," *National Cyber Summit 2016*, Huntsville, 2016, ACM.

[57] M. Ramkumar, "Minimal TCB for System-Model Execution," *The 2017 International Conference on Security and Management*, Las Vegas, 2017, number 1.

[58] M. Ramkumar, "Executing large-scale processes in a blockchain," *Journal of Capital Markets Studies*, vol. 2, no. 2, 2018, pp. 106–120.

[59] H. Saini, Y. S. Rao, and T. C. Panda, "Cyber-crimes and their impacts: A review," *International Journal of Engineering Research and Applications*, vol. 2, no. 2, 2012, pp. 202–209.

[60] D. Samyde, S. Skorobogatov, R. Anderson, and J.-J. Quisquater, "On a new way to read data from memory," *Security in Storage Workshop, 2002. Proceedings. First International IEEE*. IEEE, 2002, pp. 65–69.

[61] J. Saxon, "Spatial constraints on gerrymandering: A practical comparision of methods," 2018.

[62] M. I. Shamos and D. Hoey, "Geometric Intersection Problems," *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, vol. 1976, pp. 208–215.

[63] R. Sieber, "Public Participation Geographic Information Systems: A Literature Review and Framework," *Annals of the Association of American Geographers*, vol. 96, no. November 2004, 2006, pp. 491–507.

[64] J. P. Snyder, *Map Projections: A Working Manual*, Tech. Rep., 1987.

[65] W. Stallings, "A Brief History of Computers," *Computer Organization and Architecture Designing for Performance*, eight edition, Prentice Hall, New Jersey, New Jersey, 2007, pp. 18–20.

[66] P. Stewin and I. Bystrov, "Understanding DMA malware," *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2012, pp. 21–41.

[67] V. Thotakura and M. Ramkumar, "Minimal trusted computing base for MANET nodes," *6th International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob'2010*, Niagara Falls, ON, Canada, 2010, pp. 91–99, IEEE.

[68] J. R. Troncoso-Pastoriza, S. Katzenbeisser, M. Celik, and A. Lemma, "A Secure Multidimensional Point Inclusion Protocol," *Mm&Sec'07: Proceedings of the Multimedia & Security Workshop 2007*, Dallas, TX, USA, 2007, pp. 109–120, ACM.

[69] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," *ACM SIGARCH Computer Architecture News*. ACM, 2007, vol. 35, pp. 494–505.

[70] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger,", 2019.