Scientific Research Publishing

Journal of Information Security, 2022, 13, 101-126
https://www.scirp.org/journal/jis
ISSN Online: 2153-1242
ISSN Print: 2153-1234

Authoritative and Unbiased Responses to Geographic Queries

Mahalingam Ramkumar[1], Naresh Adhikari[2]

[1]Mississippi State University, Starkville, MS, USA
[2]Slippery Rock University, Slippery Rock, PA, USA
Email: ramkumar@cse.msstate.edu, naresh.adhikari@sru.edu

How to cite this paper: Ramkumar, M. and Adhikari, N. (2022) Authoritative and Unbiased Responses to Geographic Queries. *Journal of Information Security*, **13**, 101-126.
https://doi.org/10.4236/jis.2022.133007

Received: May 1, 2022
Accepted: June 27, 2022
Published: June 30, 2022

Abstract

A protocol for processing geographic data is proposed to guarantee authoritative and unbiased responses to geographic queries, without the need to rely on trusted third parties. The integrity of the proposed authoritative and unbiased geographic services (AUGS) protocol is guaranteed by employing novel hash tree based authenticated data structures (ADS) in conjunction with a blockchain ledger. Hash tree based ADSes are used to incrementally compute a succinct dynamic commitments to AUGS data. A blockchain ledger is used to record 1) transactions that trigger updates to AUGS data, and 2) the updated cryptographic commitments to AUGS data. Untrusted service providers are required to provide verification objects (VOs) as proof-of-correctness of their responses to AUGS queries. Anyone with access to commitments in ledger entries can verify the proof.

Keywords

Authenticated Data Structures, Blockchain Ledger, Geographic Information Systems

1. Introduction

Information systems are composed of processes for collection, organization, storage and communication of information. For information to be *authoritative* it is necessary for the receiver to establish a) the source of the information, and that b) the source does indeed have the authority to convey the information. Most often, the authority is gained through a chain of delegations. For information to be *unbiased*, it is necessary to establish that no relevant information was suppressed.

A well known example of an authoritative unbiased (AU) service is the Domain Name System (DNS) [1]. When used in conjunction with the DNS security protocol

DNSSEC [2], DNS provides AU assurances for responses to DNS queries. DNSSEC makes it possible to establish that (for example) the IP address of a domain name cs.ms.edu did indeed originate from the authoritative DNS zone ms.edu. Furthermore, in response to a query for information regarding a **non-existent** name (say) xyz.ms.edu, the response will provide *proof of nonexistence*.

In geographical information systems (GIS) [3], different types of information may be associated with various geographic objects like a point[1] (x,y), or a line segment connecting two points, or all points inside a polygonal **region**. Currently, while service providers do exist for providing useful geographic information (like one's location, restaurants and other services nearby, zip code corresponding to a location, current weather at a location, etc.), such services **can not** be considered as AU—as we are required to trust the service provider. The main contribution of this paper is a comprehensive *Authoritative and Unbiased Geographic Services* (AUGS) protocol, for guaranteeing AU responses from **untrusted service providers**, to geographic queries.

Organization

The rest of this paper is organized as follows. Section 2 provides a broad overview of the goals of AUGS. This is followed by a comparison of the goals of AUGS with that of the DNS protocol for domain names. In particular, we point out some of the unique challenges faced by AUGS due to the fact that the definition of a "geographic region" is substantially more complex than that of a "name." The former is a **list** of (x,y) coordinates on a 2-D plane, while the latter can be considered as a point on the number line. It is argued that the challenges can be overcome by using Merkle hash tree based authenticated data structures (ADS) in conjunction with a blockchain ledger.

Section 3 illustrates with examples, 2 core AUGS processes, *viz.*, **map-construction**, and **map-update**. Both processes are described in terms of simple steps involving incremental changes to AUGS data. Such simple steps are blockchain transactions that modify AUGS data. Section 4 describes a family of ADSes for computing/tracking a dynamic cryptographic commitment to different types of data. Section 5 illustrates the use of such ADSes for AUGS specific data like "list of boundary lines," "set rectangular bounding blocks," "set of points," and key-value pairs, etc. Section 5 also outlines different types of blockchain transactions that trigger incremental updates to AUGS data, and consequently, the ADS commitments. Finally, Section 5 outlines the steps involved in verifying AU properties of responses to queries.

A discussion of related works and conclusions can be found in Section 6.

2. AUGS Overview

A geographical region [4] can be described as a simple[2] n-sided polygon, where the sides of the polygon are n-boundary line segments.

[1]A point is typically specified by latitude and longitude coordinates.
[2]Sides of a simple polygon do not cross each other.

More generally, a geographic region R may be a set of simple polygons—some that enclose region R, and some that exclude region R. In Figure 1(a), regions R_2, R_3 and R_4 are represented by a one polygon each. Region R_1 is described by 3 polygons—two enclosing polygons (shaded light gray), and an exclusion polygon (darker gray region R_4). For a geographical region R defined by q_1 inclusion polygons and q_2 exclusion polygons, the region R lies to the *left* on counter clockwise (CCW) traversal of points in the q_1 polygons that enclose R (and to the **right** in q_2 polygons that exclude R).

In the rest of this paper we assume that a region is expressed as a **list of boundary lines** of the form (p_i, p_{i+1}, m) where $p_i = (x_i, y_i)$ represents point coordinates, and m uniquely identifies one of the many simple polygons of the region to which the boundary line belongs. We shall further assume that m is odd for inclusion polygons, and even for exclusion polygons.

AUGS regions may be reshaped by adding new lines. For example, by adding new lines, a polygon representing a country may be subdivided into smaller polygons representing states. Most often, the reason for creating subdivisions are for purposes of delegation of authority.

A region may be associated with different types of static and dynamic attributes like zip code, name, voting precinct, tax rates, zoning restrictions, weather information, etc. AUGS should enable anyone to

1) query such information regarding any point (x,y),

2) establish the source, and

3) establish the authority of the source (typically gained through a chain of delegations).

2.1. DNS vs AUGS

The Domain Name System (DNS) [1] is a protocol for hierarchical delegation of a name-space. A delegated DNS name is a DNS *zone*. Ownership of a DNS zone (for example) ms.edu is acquired through delegation by the parent zone edu, which was in turn acquired by delegation, from the *root* zone.

In this paper we shall use the notation $a \in b$ to signify that "a is a child of b." For example, in DNS ms.edu $\in$ edu (as ms.edu is a child of edu).

The owner of ms.edu can create any number of child names **ending** with ms.edu—for example, cs.ms.edu, ee.ms.edu, etc. Newly created names can be i) associated with different types of information—by creating different types of DNS records, or ii) **delegated** to create a new zone—by creating a special DNS record of type NS.

Analogous to a zone name in DNS, is a *region* (described by 1 or more simple polygons) in AUGS. An important difference between the DNS and AUGS is as follows:

In DNS, $x.y.z \in y.z$ as $x.y.z$ **ends with** $y.z$; in AUGS, a region $Q \in P$ if **every point** in region Q **lies inside** region P.

An AUGS region P can create any number of **non overlapping** child regions. A region P can delegate a child region (say) $Q \in P$, or associate different types of information with any point inside P, or any line segment inside P, or any polygon $Q \in P$.

2.2. AU DNS Responses

DNSSEC [2] associates every DNS zone with a public key. The public key of zone $x.y.z$ is certified by the parent zone $y.z$, whose public key is certified by *its* parent zone z, whose public key is certified by the root zone (say, Φ). The public key of the root zone is assumed to be public knowledge. In the rest of this paper, we shall use the notation $S\langle A\rangle$ for "certification of public key of A by S."

Knowledge of the root public key is sufficient to track the authority of the public key of any zone Z, by following a chain of delegations. That public key U_Z is authoritative for a DNS zone Z implies that all DNS information certified using U_Z is authoritative for names that end with Z—but only as long as such names have **not** been delegated. For example, the public key U of ms.edu is not authoritative for the name dis.cs.ms.edu if either the name cs.ms.edu or the name dis.cs.ms.edu have been delegated.

The response to a DNS query is a DNS record, along with a signature for record (signed by the zone under which the name falls), and records needed to verify the delegation chain. The steps necessary to establish AU properties of a response R_n for a name (say) n that falls in zone A where (say) $A \in B \in C$ are as follows:

1) verify certificate $A\langle R_n\rangle$ (DNS record R_n signed by zone A),

2) verify delegation chain $B\langle A\rangle$, $C\langle B\rangle$, and $\Phi\langle C\rangle$,

3) verify that i) name n ends with A; ii) name A ends with B, and iii) name B ends with C,

4) if n ends with A, but $n \neq A$ (for example, A = cs.ms.edu, but n = se.cs.ms.edu) establish that name n has **not** been delegated by A.

The last step, *viz*., proof of non-existence of delegation records—or authoritative denial—is facilitated by the NSEC/ NSEC3 [5] [6] component of DNSSEC. Authoritative denial also serves to guarantee unbiasedness of DNS. The need to prove non-existence implies that the responder (untrusted DNS servers) will not be able to (accidentally or deliberately) hide the existence of a queried DNS record.

2.3. AU Geographic Responses

Now consider the AUGS scenario with hierarchical region delegations $A \in B \in C$. The steps necessary for verifying the authoritativeness of some information I_p regarding a point $p = (x, y) \in A$ are as follows:

1) verify certificate $A\langle I_p\rangle$ for the information provided;

2) verify the delegation chain $B\langle A\rangle$, $C\langle B\rangle$, $\Phi\langle B\rangle$;

3) verify a) that $(x, y) \in A$; b) $A \in B$, and c) and $B \in C$, and

4) ensure that every point/region is uniquely delegated.

The first 2 steps (verification of certificates) are similar in both (DNS and AUGS) scenarios; both involve verification of $\mathbb{O}(1)$ certificates (does **not** depend on *N*, the total number of DNS records/AUGS data items). The third and fourth steps are however substantially different. While it is trivial for the DNS verifier to determine that "a name *a* ends with *b*" it is far from trivial in the AUGS scenario to determine if $(x, y) \in A$; it is even more difficult to determine that region $A \in B$ (or all points in region *A* fall inside region *B*). This is especially true in scenarios where regions may have tens of thousands of sides[3], may be defined by multiple polygons, and may include other delegated sub-regions.

The point location problem, *viz*., determining if a point (*x,y*) lies inside a polygon *A*, is an important problem for several application scenarios, like mouse tracking, computer graphics, GIS etc., and has attracted substantial attention in the literature [8] [9] [10] [11]. However, these algorithms were not designed with the goal of being executed as blockchain transactions. Furthermore, the problem of determining if **all** points inside a polygon *A* lie inside a polygon *B* has not attracted much attention.

Even if efficient algorithms can be designed, they can never have a complexity less than $\mathbb{O}(N)$ for a polygon with *N* sides. Unlike the DNS scenario where it is trivial for the verifier to determine that (for example) msu.edu ends with edu in the case of AUGS, the verifier will first need to access to the *N* coordinate values, and execute various point-location algorithms to establish that $(x, y) \in A$, $A \in B$, $B \in C$, etc. This is obviously impractical for an average information querier. This is the central challenge addressed by the proposed protocol, by taking advantage of i) hash tree based authenticated data structures (ADS), and ii) blockchain transactions that perform simple incremental changes to ADSes.

In the proposed AUGS protocol, verification of AU properties of any query regarding a point in a region with *N* sides will require only $\mathbb{O}(\log_2 N)$ hash operations to be performed by the verifier. More specifically, if the service provider provides information regarding *M* different regions, the size of the proof will be $\mathbb{O}(\log_2 \max(M, N))$ hashes.

The broad goal of the AUGS protocol is to permit anyone to obtain the **authoritative public key** *U* for any point (*x,y*) in the globe. Towards this end, the verifier will need the ability to verify that

1) $(x, y) \in R$, where *R* is an AUGS region;

2) (*x,y*) does **not** fall in a sub-region $R' \in R$ that has been **delegated** by *R*; and

3) *U* is the public key of region *R*.

By leveraging the public key *U*, any information from the authoritative source for point (*x,y*) can be verified.

[3]For example, in the official federal map [7], the state of Texas consists of over 600 different polygons with over 144,500 sides.

2.4. Overview of AUGS Protocol

Authenticated data structures [12]-[17] enable incremental computation of a succinct cryptographic commitment $d(t)$ to dynamic data $\mathbf{D}(t)$. Prover-verifier protocols that leverage ADSes can be used to efficiently prove specific properties[4] regarding the data $\mathbf{D}(t)$ to verifiers. Specifically, for an ADS with N data items, untrusted provers will need to provide $\mathbb{O}(\log_2 N)$ verification objects (VOs) as proof. Verifiers, who trust only the current dynamic commitment $d(t)$ to the ADS, will need to perform $\mathbb{O}(\log_2 N)$ hash operations to verify the proof. One novelty of the proposed approach is in utilizing a blockchain network to maintain universal consensus on ADS commitments.

For our purposes, a blockchain network [18] [19] [20] [21] is a distributed infrastructure to maintain an append-only blockchain ledger. Incentive mechanisms in the distributed infrastructure ensure that only *well-formed* transactions can be added to the ledger. The precise definition of "well-formed" will depend on the application. As an example, in crypto-currency applications that cater for transferring coins between crypto-wallets, a transaction to (say) "transfer n coins from wallet A to a wallet B" will be considered as well-formed only if

1) the transaction is signed by A and

2) A had n or more coins **before** the transaction was executed.

By ensuring that only well-formed transactions can be added, the integrity of current balances in all wallets is assured.

Execution of a transaction changes the **state** of the application. For crypto-currency applications, the "current state" can be described by the remaining balance in every wallet. Transferring coins from A to B changes the state of the application (wallet balances in A and B will need to be updated). In blockchain networks like Bitcoin, the state change caused by a transaction is **not** explicitly indicated in the transaction added to the ledger. On the other hand, in the Ethereum network, a cryptographic commitment to the new state(s) is made explicit in ledger entries.

In the proposed approach, processes composed of simple blockchain transactions will need to be executed by untrusted AUGS service providers to construct and update "maps." More specifically, a map *construction* process will involve converting a region represented by a list $\mathbf{L}$ of boundary lines, to a "map" $\mathbf{B}$ composed by a set of rectangular bounding boxes (BBs). A map *update* process will involve modifying the map $\mathbf{B}$ to incorporate a child region represented by a list of boundary lines $\mathbf{L}_c$ and adding "legend" entries to BBs in the map to identify points inside the sub-region.

On successful completion of such processes, an AU response to any query regarding any point (x,y) is a single BB from the set $\mathbf{B}$.

Specifically, for a region defined by N boundary lines, the service providers will need to successfully execute $\mathbb{O}(N)$ blockchain transactions that update

[4]Examples of such properties include existence of a specific item in $\mathbf{D}(t)$, or non existence of an item in $\mathbf{D}(t)$, maximum/minimum value in $\mathbf{D}(t)$, etc.

AUGS data like **L** and **B**. Such transactions will be considered well-formed only if they satisfy a **well-defined set of constraints C**. The constraints **C** ensure that untrusted service providers cannot successfully execute map construction/update processes in a way that leads to misleading/incorrect responses to queries.

The dynamic commitment to AUGS data like **L** and **B** are captured by hash tree based ADSes. The response from AUGS service providers (which is typically a BB from **B**) will be accompanied by a "proof" in the form of a small number of VOs (hashes) that can be verified against commitments in the blockchain ledger.

3. AUGS Processes

In this section we will illustrate with examples, two core AUGS processes, *viz.*, *map-construction*, and *map-update*.

3.1. Map Construction

The AUGS *map-construction* process converts a list **L** of boundary line segments representing a region, into a collection of rectangular bounding boxes (BBs). For simplicity we shall first assume a region defined by a single polygon, represented by a list $\mathbf{L} = \left[p_1 — p_2, p_2 — p_3, \cdots, p_{n-1} — p_n, p_n — p_1 \right]$. The map construction process attempts to move every boundary line in a list **L** to a rectangular BB in the collection **B** of rectangular BBs, such that the every boundary line falls wholly inside a BB. Towards this end, the process is allowed to perform simple operations like

1) split any boundary line in **L** into 2 segments (thereby increasing the total number of lines by one),

2) split any BB in **B** (horizontally or vertically, thereby increasing the total number of BBs by one), and

3) move a boundary line from the list **L** to an appropriate BB in **B**, subject to constraints **C** (to be discussed soon).

Each such operation is executed as a transaction in a blockchain network. Such transactions will be considered as **well-formed** only if they satisfy the constraints **C**, *viz.*,

C1. at most 2 boundary line segments can fall inside a BB;

C2. a single line mapped to a BB can only be a side or a diagonal of the BB;

C3. if 2 boundary line segments fall inside a BB, one should be a diagonal; the other should be an adjacent boundary line;

C4. redundant boundary points should be unique;

C5. mapping should be performed in the CCW order of points;

C6. if 2 adjacent boundary lines are mirror images of each other both lines should be removed from **L** and ignored.

Redundant boundary points are points that do not affect the enclosed region. Given two adjacent boundary lines $p_{i-1} — p_i$ and $p_i — p_{i+1}$, the point p_i is a redundant point if p_i lies on the line connecting p_{i-1} and p_{i+1}. While such

points are created by splitting boundary lines, it is possible for such points to exist even in the input (the list **L**) to the process.

Mirror images of 2 adjacent boundary lines p_{i-1}—p_i, and p_i—p_{i+1} where $p_{i-1} = p_{i+1}$ may be (as we shall see in Section 3.3.2) due to finite precision.

It is important to remember that service providers are **not** trusted, and might attempt to create invalid regions for deliberately misrepresenting information. The ability to include mirrored boundary lines and duplicate redundant points can be used by provers to misrepresent map legend entries in BBs. The 6 restrictions C1 ··· C6 ensure that the map-construction process can be successfully executed **only** if the input **L** to the process corresponds to a simple polygon. The justification for the constraints are outlined in Section 3.3.

On successful completion of the process, the collection **B** of BBs will consist of some BBs with no lines, some with one line, and some with 2 lines. More specifically, the resulting BBs can be of up to 14 types as depicted in Figure 1:

1) type 0 with no lines (clear BBs);

2) types 1 and 2 with a single diagonal line (blue BBs);

3) types 3 to 6 where the line mapped to a BB is a side of the BB (green BBs), and

4) types 7 to 14 (red BBs) where 2 adjacent boundary lines are mapped to a BB—where one is a diagonal; the shorter line (between a side and a diagonal) is associated with an offset *o*; the offset $o = 0$ if the shorter line is a BB side.

That no more than 2 lines can be mapped to a BB, and that the lines can *not* cross each other inside the BB, implies that areas within a BB can belong to at most 3 different regions, captured by 3 region legends ρ_a, ρ_b and ρ_c. Enforcing CCW order of mapping boundary lines to BBs is to ensure unambiguous labeling of regions ρ_a, ρ_b and ρ_c inside each BB. Recall that for enclosing polygons the triangular/rectangular area to the left is inside the region; for exclusion polygons the area to the right is inside the region. The legend entries ρ_a, ρ_b and ρ_c will be 0 for areas outside the region and 1 for areas inside the region.

After completion of the map-construction process, a query regarding any point (*x*,*y*) can be answered by responding with a single BB in which (*x*,*y*) falls, and examining the region markings inside the BB.

3.2. Map Construction Example

To keep the discussion simple, we shall first limit ourselves to regions defined by a single polygon. Consider a polygonal region described by a list of boundary lines {AB, BC, CD, DE, EF, FA} in Figure 2(a), bounded by a dashed rectangular bounding box (BB).

By repeated use of appropriately chosen simple operations that split a boundary line into 2 (creating a redundant boundary point), and split a BB into 2 (vertically or horizontally), we can fit every boundary line as a side or a diagonal of a BB (Figure 2(b)). For the region with 6 boundary lines in Figure 2 we

1) split boundary line AB into AE' and E'A (adding a redundant boundary

point E'), and

2) perform 8 BB splits (to result in 9 BBs marked 01 to 09), and

3) map boundary lines to BBs while strictly adhering to all constraints.

In this particular instance, 5 boundary lines become diagonals of BBs; 2 lines become a BB side; two BBs (05 and 06) are empty.

As a second example, consider a region (Figure 2(c)) PDQ. In this instance, we

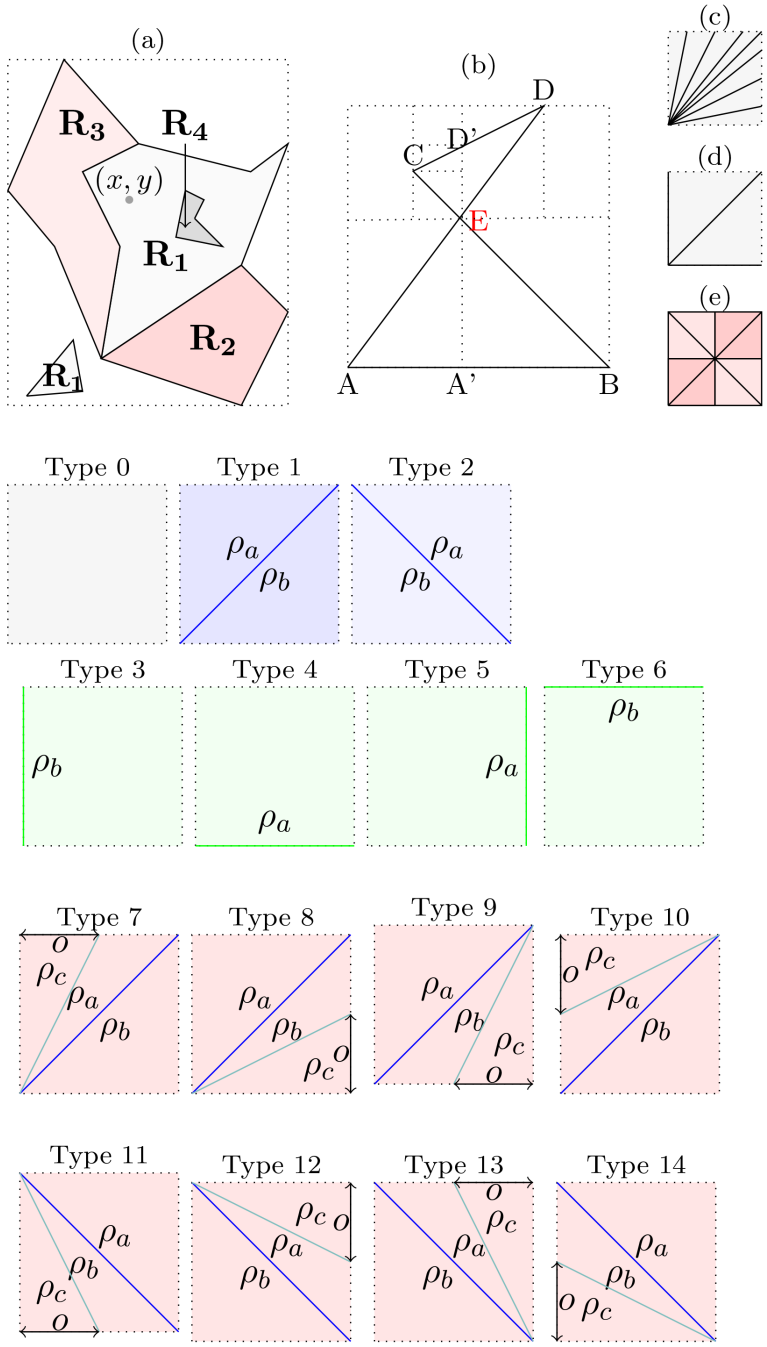

Figure 1. (a) Four geographic regions R_1, R_2 and R_3 and R_4; (b) An illegal region defined by a non-simple polygon ABCD (sides BC and DA intersect at E); (c)-(e) Effect of finite precision. Type 0 to Type 14: 15 BB types ($c = \{0 \cdots 14\}$).

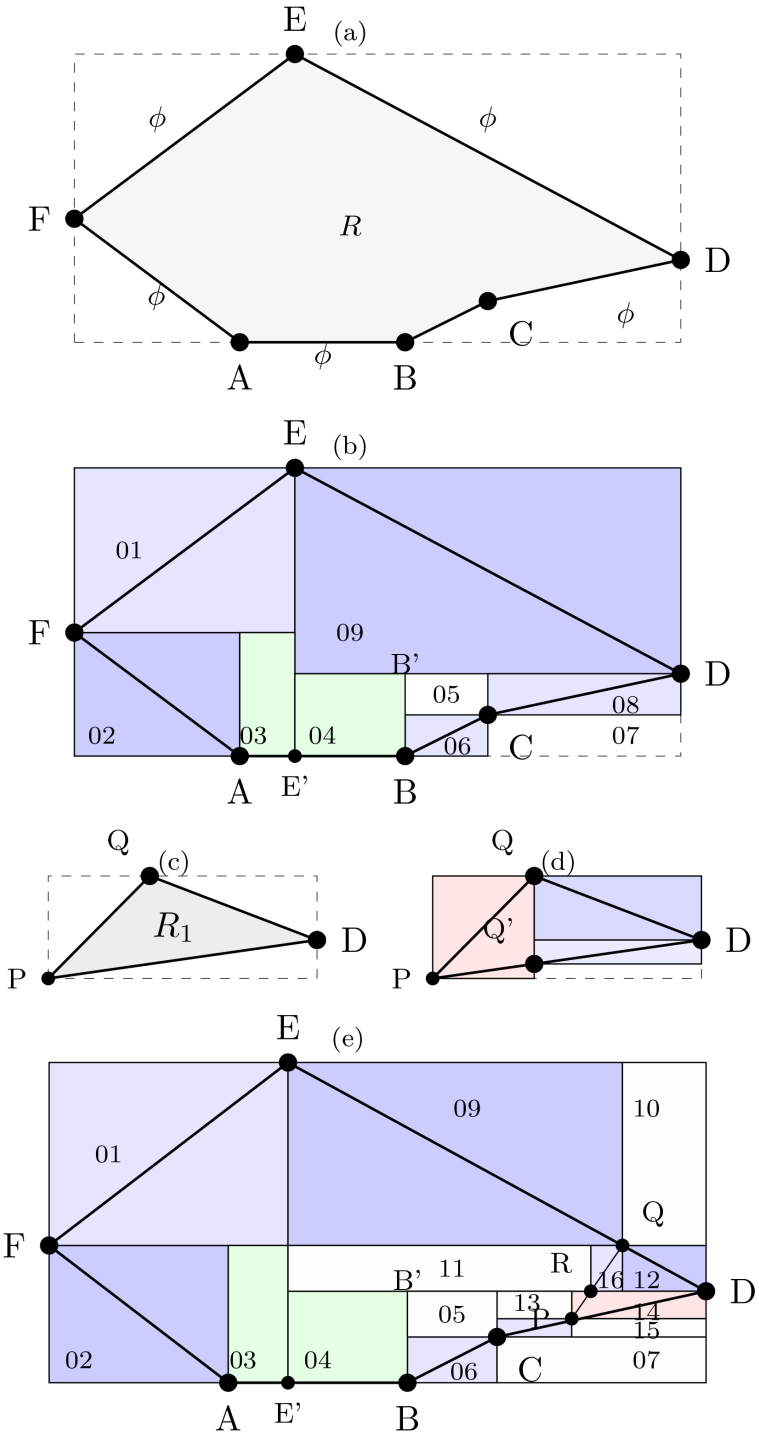

Figure 2. AUGS points and bounding boxes (BBs). (a) Parent region; (b) AUGS BBs for parent region; (c) Child region; (d) AUGS BBs for child region; (e) Altered parent region to incorporate child.

1) create a redundant boundary point Q' to split boundary line PD into PQ' and Q'D;

2) perform 3 BB splits; and

3) map boundary lines to BBs.

This time however we have to map two boundary lines (QP and PQ') to the same (red) BB (Figure 2(d)). Mapping 2 boundary lines inside a BB is unavoidable in situations where projections of adjacent boundary lines overlap in *both X* and *Y* directions.

On successful completion of the map construction process, any query regarding *any* point (*x,y*) inside the original dashed BB can be answered by examining a single BB (in the BB collection). Each BB conveys coordinates of up to 2 lines, and up to 3 region codes, from which it is straight forward to determine the region code for point (*x,y*).

For the first example the region code corresponding to line BC mapped to BB 06 in Figure 2(b) will indicate that the region above the line is 1 and the region below as 0. The red BB in Figure 2(d) will be associated with 3 region codes—0 for region below PQ', 1 for region below PQ (and above PQ'), and 0 for region above PQ.

If the query point (*x,y*) lies inside a clear BB, it is sufficient to examine an additional non-clear adjacent BB. For example, for a point (*x,y*) in BB 07, the region code is the same as the region code below the diagonal of BB 06 to the left, or below the diagonal of the blue BB 08 above.

3.3. Rationale for the Constraints

The map construction process is deemed successful only if **all** boundary lines are mapped to BBs. The 6 constraints ensure that only a sequence of points/sides representing a simple polygon can lead to successful completion of this process.

Constraint C3 ensures that boundary line segments cannot cross **inside** any BB. C4 ensures that lines do not cross at BB corners. Constraints C5 and C6 ensure unambiguous labeling of triangular/rectangular areas inside BBs.

To see the need for the constraint of uniqueness of redundant points (C4), consider the non-simple polygon represented by 4 lines {AB, BC, CD, DA} in the Figure 1(b), which obviously does *not* describe a valid region. Nevertheless, by splitting boundary lines AB at A', BC at E, CD at D', and DA at E, all resulting boundary lines (AA', A'B, BE, EC, CD', D'D, DE and EA) of this obviously invalid region can be mapped to BBs—as the boundary lines do not intersect *inside* any BB. However, the redundant points A', E, D' and E are *not* unique (as E occurs twice). This constraint will cause the map-construction process to fail.

Note that the situation would be different if the same region was described instead by a list of six boundary lines {AB, BE, ED, DC, CE, EA}. In this case only 2 redundant points will need to be added (A' and D'), to map all boundary lines. Mapping this sequence of boundary lines will be successful this time—as it should be, as the boundary lines **do** describe a valid region (two triangular regions meeting at E). While we have 2 instances of point E, they are *not* redundant points in this case.

3.3.1. CCW Mapping

The constraint of CCW order is to ensure unambiguous assignment of region

codes. Specifically, the region to the left/right of the line is labelled 1/0 for enclosing polygons (0/1 for excluding polygons). On completion of the mapping, the BB coordinates, BB type, 3 region labels, and offset o (for BB types 7 to 14 in Figure 1) are sufficient to unambiguously determine the precise coordinates of lines mapped to the BB, and consequently, determine the region legend for point (x,y) (1 for inside the region and 0 for outside). To ensure that the mapping is sequential it is verified that the next line starts where the last mapped line ended. To ensure that the order is CCW (and not CW) the area of the polygon is incrementally computed using the shoe-lace formula [22]. As every line is mapped sequentially, only CCW mapping will result in a positive value for the area (CW mapping will result in a negative value).

The "shoelace area" A is initially set to 0; if a line segment (x_1,y_1)—(x_2,y_2) is added to the map the shoelace area is incremented by $\delta_A = x_1y_2 - x_2y_1$. On completion of the mapping, the cumulative shoelace area[5] A will be positive only if the traversal in performed in CCW order. If $A<0$ the process is considered to have failed (the ill-formed transaction will be rejected).

3.3.2. Implications of Finite Precision

There are obviously scenarios where it is necessary to map 2 lines to a BB (when 2 adjacent boundary lines have overlapping projections in *both* X and Y directions). Fortunately, even for maps with multiple polygons 2 lines to a BB is sufficient.

Consider a scenario in Figure 1(c) where several boundary lines meet at a corner of a BB. It might appear at first glance that how many every subdivisions are performed to split boundary lines or BBs, a BB at the lower left corner, at which all lines meet, will still have more than 2 lines mapped to it.

In practice, however, only a finite number of bits can be used to represent x and y coordinates. For example, representing coordinates using unsigned 32-bits is sufficient to realize a worst case resolution error (at the equator) of about 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 (Figure 1(d)). 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 (Figure 1(e)).

An important consequence of finite precision is in scenarios where boundary lines meet at a point (say p_2) at highly acute angles. Let the two adjacent lines p_1—p_2, p_2—p_3 fall inside the same BB, where p_1 and p_3 have the same x-coordinate (where they meet a BB side). If the y coordinates of the two points differ by less than half the smallest BB resolution, then p_1 and p_3 will be quantized to the same point $p_1 \approx p_3 \approx p'$, thereby creating two adjacent boun-

[5] $A/2$ is the actual area of the polygon.

dary lines $p \text{—} p'$ and $p' \text{—} p$, that mirror each other, creating a zero-area region between them. Mapping such lines will result in inconsistent region codes. Fortunately, removing such lines (and not mapping them) will have no practical effect on the geometry of the region. Such adjacent pairs, if present (either due to precision issues, or deliberately added by untrusted provers), will be ignored.

3.4. Map Update

The *map-update* process modifies the map of a region (created by a map-construction process) to include a sub-region. This process may need to split BB's created by the map-construction process further, add new lines inside BBs, and update legend labels for areas inside BBs. While region label 1 is used for the region created by the map construction process, region codes 2 and higher are used for labeling sub-regions created by the map-update process. This process can be performed any number of times to create any number of non overlapping child regions, with unique region legend labels.

Consider a scenario in **Figure 2(e)** where the polygon PDQ in **Figure 2(c)** is actually a child of region ABCDEFA in **Figure 2(a)**. In this specific case, by further splitting BBs 09 and 08 in **Figure 2(b)** as in **Figure 2(e)** we can incorporate the second region too in the map of the first region. More generally, the map update process can be seen as consisting of the following steps:

1) Move some existing lines from some BBs in the parent map to a *temporary list*; in this case lines ED (BB 09 in **Figure 2(b)**) and CD (BB 08 in **Figure 2(b)**) will be moved to the temporary list.

2) Split (now empty) BBs as necessary; in this case BB 08 in **Figure 2(b)** was split into 4 BBs (08, 13, 14 and 15 in **Figure 2(e)**); BB 09 in **Figure 2(b)** was split into 5 BBs (09, 10, 11, 16, 12 in **Figure 2(e)**)

3) Split lines in the temporary list as necessary; line CD was split into CP and PD; line ED was split into EQ and QD.

4) Map all lines from the temporary list back to BBs; 4 lines CP, PD, DQ and QE will be re-mapped (to BBs 08, 14, 12 and 09 respectively, in **Figure 2(e)**).

5) Split boundary lines of the child polygon as necessary; QP was split into QR and RP.

6) Map all child boundary lines to the altered parent's map, one by one, in CCW order, and update region codes as necessary.

If the legend index assigned to the child is 2, on completion of the map update process the region codes in BBs 12 will be $\rho_a = 0$, $\rho_b = 2$ (region ρ_c not present in blue BBs). In BB 16 the region code will be $\rho_a = 1$, $\rho_b = 2$; in BB 14 the region codes will be $\rho_a = 2$, $\rho_b = 0$, $\rho_c = 1$.

As the child boundary lines are mapped to BBs of the parent, the region codes to the left of the line (or right of the line for island polygons) are modified from "1" to the unique legend label (for example, 2) assigned to the child.

As mentioned earlier the list of boundary lines **L** and set of BBs **B** are hash tree based ADSes outlined in the next section.

4. Authenticated Data Structures

Given a cryptograpic hash function $h()$, and

$$y = h(x), x \in \{0,1\}^*, y \in \{0,1\}^n, \quad (1)$$

it is safe to conclude that "pre-image x **existed before** hash y." This is due to the fact that *if* y were chosen first it is computationally impractical to come up with a suitable pre-image.

Cryptographic hash functions lie at the core of several hash tree based ADSes that enable incremental computation of a succinct cryptographic commitment $d(t)$ to dynamic data $\mathbf{D}(t)$.

In prover-verifier protocols that leverage such ADSes, irrespective of the number of items N in dynamic data $\mathbf{D}(t)$, it is sufficient for the verifiers to track a single dynamic hash $d(t)$. Provers (who actually store all data $\mathbf{D}(t)$) can prove specific properties of the $\mathbf{D}(t)$ to verifiers by providing $\mathbb{O}(\log_2 N)$ VOs. The verifiers, *who do not trust the prover*, can verify the proof by performing $\mathbb{O}(\log_2 N)$ hash operations and comparing the result with the current commitment $d(t)$.

4.1. Ordered Merkle Trees

A binary Merkle hash tree [23] of depth d can have up to $N = 2^d$ leaves and $2N-1 = \sum_{i=0}^{d} N/2^{d-i}$ nodes. Each leaf is a data record. The N leaf-nodes at depth d are leaf hashes; the $N/2$ nodes at depth $d-1$ are obtained by hashing 2 nodes in depth d together, and so on. The 2 nodes at depth 1, are finally hashed together to result in a lone node—the root of the tree—at depth 0. The root is a commitment to all leaves and nodes.

Corresponding to any node v at depth k are k complementary nodes $c_1 \cdots c_k$—one each at depths $k, k-1, \cdots, 1$. The complementary nodes are VOs that permit any entity with access only to the root $v_{00} = \xi$ of the tree to verify existence of node v, at depth k, in a tree with root ξ. Proof of existence of a leaf-node is proof of existence of its preimage, *viz.*, a leaf. By performing $\log_2 N$ hash operations using $\log_2 N$ VOs, a verifier with access to the root can verify the existence of any leaf in the tree, and/or compute the new root corresponding to an incremental update like a) modification to a leaf, or b) insertion/deletion of a leaf.

Unfortunately, the Merkle hash tree does not enable the prover to efficiently prove **absence** of a specific leaf/record. This limitation is addressed by ordered Merkle trees (OMT) [14] [15] which imposes well defined rules for insertion/deletion of leaves. The rules for insertion/deletion guarantee *completeness of the collection* represented by leaves of the tree, at all times.

Each leaf in an OMT corresponds to an *interval on the number line*. An interval/leaf $[a, b[$ represents

1) (if $a < b$): all $a \le x < b$;

2) (if $b \le a$): all $x \le a$ and all $x > b$ (note that if $b = a$ the interval

represents the entire number line $x \le a$ and all $x > a$).

Leaves can be inserted only by splitting an interval into 2—thereby, replacing a leaf with 2 leaves. Merging 2 adjacent intervals results in removal of a leaf. These restrictions ensure that leaves of an OMT form a complete collection at **all times**. More specifically, a unique leaf will exist for *every* point on the number line.

The first leaf inserted into an empty OMT will correspond to an interval $[x, x[$ (all values greater than equal to x and all values less than x). To create a new interval from x to y where $y > x$ we can split the leaf $[x, x[$ into 2—one corresponding to the desired interval $[x, y[$ and the second corresponding to a wrapped around interval $[y, x[$. As an example, the 3 leaves $[2, 9[$, $[9, 22[$ and $[22, 2[$ for a complete collection.

An OMT can also be interpreted as a key-value collection. With this interpretation, the start k of an interval $[k, k'[$ is seen as a unique key. Under this interpretation, existence of a leaf $([k, k'[, v)$ in a hash tree with root ξ implies

1) existence of a key-value pair $\{k, v\}$; and

2) that the next-key is k' implying *non-existence* of all other keys that fall in the interval. Note that *if* $x > x'$, then x and x' are the highest and lowest keys (respectively).

Thus, by demonstrating the existence of a leaf, the prover can now demonstrate **non-existence** of specific keys, highest/lowest keys, etc. Elementary OMT operations like insertion, deletion, and leaf updates have the same $\mathbb{O}(\log_2 N)$ complexity as Merkle tree operations.

4.2. 2-D OMT

One important contribution of this paper is a 2-D extension of OMTs. A 2-D OMT can be seen as a complete collection of 2-D intervals (or *bounding boxes*) of the form $[(x, y), (x', y')[$. Specifically, $[(x, y), (x', y')[$ represents a planar rectangular region extending between $[x, x'[$ in one dimension and between $[y, y'[$ in the second dimension. A collection with a single BB will specify a wrapped around interval $[(x, y), (x, y)[$ representing the entire 2-D plane.

The first leaf (BB) inserted into an empty 2-D OMT should be of the form $[(x, y), (x, y)[$, (representing the entire 2-D plane). A leaf for interval (say) $[(2,3), (5,6)[$ can be split into two (for example)

1) horizontally as $[(2,3), (5, y)[$ and $[(2, y), (5,6)[$, where $3 < y < 6$, or

2) vertically as $[(2,3), (x,6)[$ and $[(x,3), (5,6)[$, where $2 < x < 5$.

A leaf for a wrapped around interval $[(9,6), (5,8)[$ may be split

1) along a vertical into $[(9,6), (x,8)[$ and $[(x,6), (5,8)[$ where $x > 9$ or $x < 5$; or

2) horizontally as $[(9,6), (5, y)[$ and $[(9, y), (5,8)[$ where $6 < y < 8$.

For our purposes, leaves of a 2-D OMT are five-tuples that convey the four values x, y, x', y' (BB coordinates), and a value v associated with the BB. A 2-D OMT is guaranteed to include a leaf for *every* point (x, y) on the plane. More

specifically, a query regarding *any* point (x, y) on a plane can be answered by providing a single leaf from a 2-D OMT (along with $\log_2 N$ VOs, where N is the total number of BBs).

5. Process States and Transactions

AUGS employs a variety of Merkle tree and OMT based data structures to compute dynamic commits to AUGS process states.

5.1. Process States

AUGS process states that are incrementally updated by AUGS transactions are as follows:

1) region data: leaves of a Merkle tree with commitment ξ_r;
2) map data: leaves of a 2-D OMT (BBs) with commitment ξ_b;
3) redundant-points: a key-value collection with commitment ξ_p; and
4) temporary-lines: leaves of a Merkle tree with commitment ξ_t.

For simplicity of notations, in the rest of this paper, we will use the following conventions:

1) A collection (leaves of a hash tree with root ξ) will simply be referred to as "collection ξ."
2) $L \in \xi$ implies "a leaf L in collection ξ."
3) $k \notin \xi$ indicates "key k does not exist in collection ξ."

5.1.1. AUGS Regions

An AUGS region (composed of one or more polygons) consists of boundary lines of the form (p_1, p_2, m) where i) m is a polygon identifier; and ii) $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ are the start and end points of a boundary line (when the polygon is traversed in the CCW order). The polygon identifier m is odd for polygons that enclose the region and even for polygons that exclude the region.

Each boundary line is a leaf of a hash tree with root ξ_r. A region described by u polygons, each with $n_1, n_2, \cdots, n_u$ sides respectively, is a tree with $\sum_{i=1}^{u} n_i$ leaves.

Transaction SplitLine (p_1, p_2, m, p) splits a line $(p_1, p_2, m) \in \xi_r$ into (p_1, p, m) and (p, p_2, m), only if p lies on $p_1 — p_2$. This operation replaces one leaf with 2.

Transaction SkipLines (p_1, p_2, m) *removes* two adjacent boundary lines $(p_1, p_2, m) \in \xi_r$ and $(p_2, p_1, m) \in \xi_r$ that are mirror images.

5.1.2. AUGS "Map"

An AUGS map is a complete 2-D OMT collection with root ξ_b. A leaf $\mathbf{b} = (x_l, y_l, x_h, y_h, v)$ in this tree represents a rectangular bounding box (BB). The value v assigned to a BB depends on the line(s) mapped to the BB, and is of the form

$$v = c \,\|\, \rho_a \,\|\, \rho_b \,\|\, \rho_c \,\|\, o, \tag{2}$$

where c is the BB type, ρ_a, ρ_b, ρ_c are region codes, and o is an offset.

Transaction SplitBB() splits an **empty** BB in ξ_b into 2 (vertically or horizontally). If $\xi_b = 0$, then this transaction sets $\xi_b = h(x, y, x, y, 0)$ corresponding to a lone BB that spans the entire 2-D plane.

5.1.3. Mapping Lines and Redundant Points

The hash tree with root ξ_p is a key-value collection where each key corresponds to a (redundant) point, and the value is unused (and thus, omitted). Leaves of this tree are of the form (k, k') where $k = x \| y$, $k' = x' \| y'$. Points determined to be redundant are added to this collection by a transaction MapLine(). This collection is used to guarantee uniqueness of redundant points (as keys have to be unique).

Transaction MapLine($p_1, p_2, m, \mathbf{b}$) i) removes a boundary line $(p_1, p_2, m) \in \xi_r$ and maps it to a BB in $\mathbf{b} \in \xi_b$ by updating the value v of the BB and ii) *if* p_1 is determined to be a redundant point, adds key p_1 to collection ξ_p. Transaction MapLine will be ill-formed if the redundant point **already** exists in ξ_p.

5.1.4. AUGS Temporary Boundary Lines

The hash tree with root ξ_t is used to temporarily store some boundary lines during the map update process. Leaves of this tree are of the form $(p_1, p_2, \rho_a \| \rho_b)$ and convey region codes above and below the line $p_1 — p_2$ (for vertical lines ρ_a is the region code to the left).

Transaction UnmapLine() removes (up to 2) existing lines from a BB $\mathbf{b} \in \xi_b$, sets the value of the BB $\mathbf{b}$ to indicate BB type 0, and adds the line(s) to the tree with root ξ_t.

Transaction SplitTmpline(p_1, p_2, p) splits a line $(p_1, p_2, u = \rho_a \| \rho_b)$ into (p_1, p, u) and (p, p_2, u) (only if p lies on the line segment $p_1 — p_2$).

Transaction RemapLine() removes a line from $(p_1, p_2, \rho_a \| \rho_b) \in \xi_t$ and maps it back to a BB in ξ_b (and updates the BB type and region codes).

5.2. Macro-Transaction ConstructMap()

The state of a map construction process is captured by seven dynamic values, *viz.*,

$$\mathcal{S}_{mc} = \{\xi_r, \xi_b, \xi_p, p_s, p_b, p_e, A\}, \tag{3}$$

where ξ_r is the commitment to a region (with any number of polygons), ξ_b is the commitment to a map, ξ_p is a commitment to a collection of unique redundant points.

Tracking the 3 point values p_s, p_e, p_b, and the "shoelace area" A, makes it possible to i) identify redundant points and mirrored boundary lines ii) ensure that all polygons in ξ_r are closed; and iii) ensure that traversal was performed in the CCW order. All 4 values are affected by the MapLine ($p_1, p_2, m, \mathbf{b}$) transaction.

The shoelace area A is initially set to 0. As a line $p_1 — p_2$ is added to the map

where $p_1.x$, $p_1.y$, $p_2.x$ and $p_2.y$ are the respective coordinates of p_1 and p_2, the elemental shoelace area is

$$f_{sl}(p_1, p_2) = p_1.x\,p_2.y - p_2.x\,p_1.y \tag{4}$$

On completion of the mapping the cumulative shoelace area[6] A will be positive only if the traversal in performed in CCW order.

If $p_s = 0$ the implication is that the line (p_1, p_2, m) to be mapped is the first line of polygon m in ξ_r. Consequently, p_s is initialized to p_1 (and will be reset to 0 when a line (p, p_s, m) is mapped). In addition, all leaves in the tree ξ_p (redundant points) are removed by setting $\xi_p = 0$. The values p_b and p_e are set to p_1 and p_2 (end points of the most recently mapped line).

If $p_s \neq 0$, MapLine $(p_1, p_2, m, \mathbf{b})$ expects $p_e = p_1$, and $p_2 \neq p_b$. Recall that scenarios (involving two adjacent lines of the form (p_1, p_2, m) and (p_2, p_1, m)) can only be handled using transaction SkipLines. If p_1 lies on the line $p_b — p_2$, p_1 is a redundant point, and should be added as a key to ξ_p. Inability to do so (if p_1 already exists in ξ_p) will result in failure of the transaction. If $p_2 = p_s$ MapLine recognizes end of the current polygon and sets $p_s = 0$. The pseudo-code for transaction MapLine() is depicted in Figure 3.

```
MapLine(p1, p2, m, b)
  IF line not in a valid BB type RETURN ERROR
  IF ps == 0
    ps = p1, pb = p1, pe = p2, ξp = 0, A = A + fsl(p1, p2)
  IF ps ≠ 0
    IF p1 ≠ pe RETURN ERROR
    IF p2 == pb RETURN ERROR
    pb = p1, pe = p2, A = A + fsl(p1, p2)
    IF p1 lies on pb–p2
      IF p1 ∈ ξp RETURN ERROR
      ELSE insertkey(p1, ξp)
    IF p2 == ps
      ps = 0, A = A + fsl(p1, p2)
  Update region codes in BB b
  Update ξr, ξb
```

Figure 3. Micro-transaction MapLine().

```
MapChildline(p1, p2, m, b)
  IF line not in a valid BB type RETURN ERROR
  IF ps == 0
    ps = p1, pe = p2
  IF ps ≠ 0
    IF p1 ≠ pe RETURN ERROR
      pe = p2
    IF p2 == ps
      ps = 0
  Update region codes in BBb
  Update ξr, ξb
```

Figure 4. Micro transaction MapChildline().

[6] $A/2$ is the actual area of the polygon.

The map construction process can be seen as a macro-transaction, consisting of a sequence of micro transactions of 4 types: *viz*., SplitLine, SplitBB, MapLine, and SkipLines().

The macro transaction **ConstructMap** (ξ, v_r, v_b) constructs a map (with commitment) v_b for a region (with commitment) v_r, by performing a sequence of micro-transactions specified as leaves of a hash tree with root ξ. The commitment $v_r = R$ to the region (root of a Merkle tree with boundary lines as leaves) also serves as a unique region identifier.

The initial state of the process is set to

$$\mathcal{S}_{mc} = \left\{\xi_r = v_r, \xi_b = \xi_p = p_s = p_b = p_e = A = 0\right\}. \tag{5}$$

From this point onwards, the micro-transactions in the Merkle tree with root ξ are executed sequentially to split lines in ξ_r, split BBs in ξ_b, remove mirrored lines from ξ_r, and map lines in CCW order to ξ_b. At the end of the process, $\xi_r = 0$ (implying that all lines have been mapped) and $p_s = 0$ (the last polygon is closed). In other words, the macro-transaction is well-formed if

1) every micro-transaction in ξ is well-formed, and

2) the final state is

$$\mathcal{S}_{mc} = \left\{\xi_r = 0, \xi_b = v_b, \xi_p, p_s = 0, p_b, p_e, A > 0\right\}. \tag{6}$$

On successful completion of the process the region $R = v_r$ is determined to be valid, and is associated with a map (with root) ξ_b.

5.3. Macro-Transaction UpdateMap()

The state of the map update process is captured by five values, *viz*,

$$\mathcal{S}_{mc} = \left\{\xi_r, \xi_b, \xi_t, p_s, p_e\right\}, \tag{7}$$

where ξ_r is the commitment to a child region (which has already undergone a map-construction process), ξ_b is the commitment to the map of the parent region, and ξ_t is a commitment to a collection temporary lines. As earlier, p_s is the first mapped point of a polygon in ξ_r, and p_e is the end point of the previously mapped boundary line.

UpdateMap $(\xi, v_r, v_b^0, v_b^f, n)$ modifies a map v_b^0 to v_b^f to incorporate a child region v_r, using a sequence of micro-transactions in a hash tree with root ξ. The value n is legend label to be assigned to the child region.

Seven types of micro-transactions are used by the map update process: SplitLine, SplitBB, SkipLines(), UnmapLine, SplitTmpLine, RemapLine, and MapChildline. The initial state of the process is

$$\mathcal{S}_{mu} = \left\{\xi_r = v_r, \xi_b = v_b^0, 0, 0\right\}, \tag{8}$$

From now on, the micro-transactions remove lines from ξ_b and map them to ξ_t, split lines in ξ_t as necessary, split BBs in ξ_b, and finally, map lines in ξ_r to ξ_b, in CCW order. At the end of the process $\xi_r = 0$, implying that all lines have been mapped. The micro-transactions also update values p_s, p_b, p_e to ensure that lines are mapped in the CCW order.

Transaction MapLine (used in the construction process) is intended for boundary lines of regions that have *not yet* been verified to be valid. On the other hand, boundary lines of a child region mapped by MapChildline have already been confirmed to correspond to a valid region (by a prior successful execution of the map-construction process on the child region). Consequently, MapChildline does not need to check for uniqueness of redundant points, or that the points are arranged in the CCW order. The pseudo code for this transaction is shown in **Figure 4**.

The macro-transaction is well-formed if every micro-transaction in ξ is well-formed, and execution of all micro-transactions results in final state

$$\mathcal{S}_{mu} = \left\{\xi_r = 0, \xi_b = v_b^f, \xi_t = 0, p_s = 0, p_e\right\}. \tag{9}$$

On successful completion of the process on a existing map of a region *R*, a region $R' = v_r$ is recognized a genuine child of region *R* with legend *n* in the map of region *R*. The commitment to the modified map of *R* is updated to v_b^f.

5.4. Global AUGS States

The state of all AUGS data for any number of regions is captured by 4 key-value collections that specify relationships between regions, maps, parent-child relationships, and public keys of regions. The collections include

1) static region collection with root r_s : a key-value pair $\{R, v_b\}$ in this collection implies that *R* has been verified to be a valid region (defined by 1 or more simple polygons).

2) dynamic region collection with root r_d : $\{R, v_b \| n\} \in r_d$ indicates that the region R has $n-1$ sub-regions with legend entries 2 to *n*.

3) parent-child collection with root r_c : $\{R, R_p \| n\} \in r_c$ indicates that region *R* is the child of a parent R_P with legend entry *n* in the map of parent R_P;

4) public key collection with root r_u : $\{R, U\} \in r_u$ conveys that *U* is the authorized public key for every point in region *R*.

The broad goal of the AUGS protocol is to permit anyone to obtain the authoritative public key *U* for any point (*x,y*) in the globe. Using the public key, any information from the authoritative source can be verified.

AUGS ledger entries provide a record of transactions, that include commitments to AUGS states after each AUGS transaction. While micro-transactions for a specific region updates up to 7 temporary region-specific states in the ConstructMap (or 5 temporary region-specific states for UpdateMap) process, successful execution of macro transactions result in updates to the 4 AUGS states r_s, r_d, r_c and r_u.

Successful execution of **ConstructMap** (ξ, v_r, v_b) for a region with root $R = v_r$ resulting in a map with root v_b, is proof that the list of boundary lines with commitment v_r constitute a valid region (described by 1 or more simple polygons). The end result is addition of a key-value pair $\{R, v_b\}$ to the static region collection (which updates state r_s).

Before the map v_b of a region *R* can be updated, it is necessary to assign a

public key for region R, as updates to the region R will need to be authorized. This process will be discussed shortly, in Section 5.5.

Execution of **UpdateMap** $(\xi, R, v_b^0, v_b^f, n, R_p, U_p, \Sigma_p)$ for updating the map of a parent R_P to incorporate a child region R with legend entry n in the map of parent R_P will commence only if

1) $\{R_p, v_b^0 \| n-1\} \in r_d$, indicating $n-1$ as the highest legend entry;

2) $\{R_p, U_p\} \in r_u$ indicating that the region R_P has the authority to create it's own child regions;

3) the signature Σ_P for the transaction is verifiable using public key U_P.

On successful execution of **UpdateMap** $(\xi, R, v_b^0, v_b^f, n, R_p, U_p, \Sigma_p)$ for updating the dynamic map of parent region R_P

1) $\{R_p, v_b^0 \| n-1\} \in r_d$ is updated to $\{R_p, v_b^f \| n\}$;

2) $\{R, R_p \| n\}$ is added to the parent-child collection to convey that R is a valid child of region R_P, and is associated with region legend n in the dynamic map of the parent.

A successful **UpdateMap** $(\xi, R, v_b^0, v_b^f, n)$ transaction results in updates to AUGS states r_d and r_c.

5.5. Certifying Public Keys of Regions

Just as a DNS zone A can authenticate the public key of child zone $B \in A$ (or name B ends with A), AUGS regions can certify public keys of child regions to delegate responsibility of authenticating responses regarding points inside the child zone. Public key certificates are added to the key-value collection with root r_u.

Specifically, the public-key collection has key-value pairs of the form $\{R, U\}$. A transaction **CertifyPK** $(R_p, R_c, v_c, n, U_c, U_p, \Sigma_p)$ can be invoked by parent region R_P (with public key U_P) to certify the public key U_c of child region R_c. This transaction is deemed to be well-formed only if

1) $\{R_p, U_p\}$ exists in public key collection;

2) key R_c does **not** exist in public key collection r_u;

3) key R_c does **not** exist in the dynamic region collection r_d;

4) $\{R_c, v_c\}$ exists in the static region collection r_s;

5) $\{R_c, R_p \| n\}$ exists in the parent-child collection r_c; and

6) signature Σ_p for the transaction is consistent with public key U_P of the parent region R_P.

This transaction results in

1) addition of a key-value pair $\{R_c, U_c\}$ to the public key collection r_u;

2) addition of a key-value pair $\{R_c, v_c \| 0\}$ to the dynamic region collection r_d; from this point on, the map of child region R_c may be updated by the child to create sub-regions inside R_c.

A transaction **RootCertify** (R, v_b, U, Σ_ϕ) is used to assign a public key U to a region R that does **not** have a parent region. The public key is signed by the root Φ.

This transaction verifies that key R does **not** exist in the parent-child collection or dynamic collection, but $\{R, v_b\}$ **does** exist in the static collection (indicating that R is a valid region). Furthermore, the signature for the transaction should be consistent with the root public key.

The root public key is initialized as the first entry in the public key collection, as a key value pair $\{\Phi, U_\phi\}$ corresponding to a special region identifier (say, $\Phi = 1$). The result of this transaction is

1) addition of a key-value pair $\{R, U\}$ to the public key collection;

2) addition of a key-value pair $\{R, v_b \| 0\}$ to the dynamic region collection;

3) addition of a key-value pair $\{R, \Phi\}$ to the parent-child collection to convey the creation of a special region by the root Φ.

5.6. Verifying Responses

Reliable current values of the dynamic AUGS states r_s, r_d, r_c and r_u can be obtained by any verifier at any time from the blockchain network. Armed with this, one can proceed to obtain the authoritative public key U for any point (x,y).

To prove AU properties of the response "U is the public key for point (x,y)," the response includes

1) a key-value pair $\{R, v_b \| n\}$ from the dynamic region collection (along with $\log_2 M$ VOs, where M is the number of AUGS regions) with root r_d;

2) a leaf (BB) from a BB collection with root v_b (along with $\log_2 N$ VOs, where N is the number of BBs in the map of region R);

3) a key-value pair $\{R, U\}$ from the collection with root r_u (along with $\log_2 M$ VOs).

4) additionally, **if** the legend corresponding to point $(x, y) = m > 1$ in the BB, a key-value pair $\{R_c, R \| m\}$ from the parent-child collection and **non existence** proof of key R_c in the public key collection are also required.

Having verified the first 2 key-value pairs and the BB, using the VOs, the verifier can proceed to verify that (x,y) does indeed fall inside the rectangle (BB), and that the region code for the (x,y) is 1.

Recall that each BB specifies the enclosing x and y coordinates, and a value v which indicates the BB type, offset o (for types 7-14), up to three region codes for triangular/rectangular areas inside the BB. If the region code is 0, then (x,y) is **not** a point in region R. The response will not be accepted as AU.

Similarly, if the region code is $m > 1$, then (x,y) is a point inside a sub-region R_c of R (as indicated by the key-value pair $\{R_c, R \| m\}$). However, the proof of non-existence of a public key for the child region R_c is proof that the region has not been delegated. In other words, the public key U of the parent R still remains authoritative for the sub-region R_c with legend m.

Finally, if U is deemed authoritative for point (x,y), the verifier can then accept any information regarding point (x,y), as long as it is duly authenticated using a signature verifiable using public key U.

6. Related Work

Several hash tree based authenticated data structures (ADS) [12]-[17] have been proposed in the literature which enables an untrusted prover to prove a wide variety of properties regarding a wide variety of data to verifiers who have access only to the succinct commitment for all data. Most often, in application scenarios relying on ADSes, the creator/owner of the data constructs a suitable ADS, and disseminates the commitment to the tree by signing the commitment (root of the tree). An important novelty in AUGS protocol is that of combining the utility of ADSes with the power of blockchain networks to maintain consensus on the dynamic commitment at all times.

A precursor to the AUGS protocol is the secure queryable dynamic maps (SQDM) [24] protocol. Both SQDM and AUGS rely on simple state-change functions that convert a vector representation of a region (a sequence of boundary lines or points) to a map representation. The main differences between SQDM and AUGS are three-fold.

The first difference lies in the data structures used for capturing succinct commitments to regions and maps. SQDM [24] employed nested 1-D OMTs where two-dimensional space was first split into vertical bars and each vertical bar was then split into multiple BBs. Compared to the 2-D BBs in AUGS, the SQDM approach calls for a substantially larger number of BBs to represent a map. The second is the strategy for ensuring correctness of state-change functions; SQDM relied on trusted hardware (instead of blockchain). Thirdly, SQDM was intended to merely guarantee integrity of responses to point-location [8], [11] queries. SQDM did not have the ability to cater for delegated sub-regions.

7. Conclusions

AUGS is a comprehensive protocol that can serve as a foundation for any application where information is tied to geography. Just as the domain name system [1] permits authenticated and unbiased responses to queries regarding names and types, AUGS caters for AU responses to queries regarding any point (x,y) in a 2-D plane. More specifically, the DNS security protocol DNSSEC [2] permits any one to determine the authoritative public key U_Z of the zone Z under which the queried name A falls. This public key U_Z can then be used to verify authenticity of specific types of information regarding the name A. In AUGS, the response to any query regarding (x,y) is the public key U_R of the region R in which point (x,y) falls. The main challenge addressed by AUGS is that while determining that "name a.b.c belongs to zone b.c" is trivial, it is far from trivial to determine that (x,y) falls inside a *non-delegated* portion of region R, where a region R may be described by one or more polygons with possibly tens of thousands of sides, and that any number of non-overlapping polygonal sub-regions inside region R could have been delegated.

That AUGS guarantees **authoritative** geographic information implies that a wide range of government services can be moved completely to the digital realm.

Just as one can buy/surrender a domain name by performing purely digital transactions, one can a) buy/sell land parcels or farms, b) assume control over a region for a specific purpose, c) propagate authoritative information associated with the region (for example, information necessary for emergency responders [25], local laws, tax rates, roadwork in progress), d) delegate authority over sub-regions to other entities for specific purposes (for example, a utility district serviced by a power company), etc.

That AUGS can guarantee **unbiased** information regarding availability of different types of services at/near specific points/regions implies that a wide range of commercial (location-sensitive) services can be readily and reliably provided/discovered by any one, without the need to trust middle-men.

In practice, the several public keys may be authoritative for a point (x,y) depending on the **context** of the query. For example, the **authority** for responses to different queries like a) current weather at (x,y) or b) the polling location for a resident at (x,y) or c) the prevailing local tax-rate at (x,y) or d) zoning restrictions at (x,y) or e) the zip code of (x,y) or f) the owner of (x,y) or g) utility pipes buried near (x,y), or h) hazardous material storage location near (x,y), will be different. One way to cater for different contexts is to maintain independent AUGS infrastructures for each context. However, a single infrastructure capable of supporting multiple contexts can be more efficient. Such context specific additions to AUGS are one of our ongoing research topics.

Acknowledgements

This research was partially funded by the United States Department of Agriculture, Agricultural Research Service (USDA-ARS): 58-0200-0-002, "Advancing Agricultural Research through High Performance Computing."

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Mockapetris, P.V. (1987) Domain Names—Concepts and Facilities. RFC Editor. https://doi.org/10.17487/rfc1034

[2] Arends, R., Austein, R., Larson, M., Massey, D. and Rose, S. (2005) RFC 4033: DNS Security Introduction and Requirements. https://doi.org/10.17487/rfc4033

[3] Chang, K. and Tsung, K. (2016) Introduction to Geographic Information Systems. 9th Edition, McGraw-Hill, New York.

[4] ESRI White Paper (1998) ESRI Shapefile Technical Description. https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[5] Weiler, S. and Ihren, J. (2006) RFC 4470: Minimally Covering NSEC Records and DNSSEC On-Line Signing. https://doi.org/10.17487/rfc4470

[6] Laurie, B., *et al.* (2008) DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155. https://doi.org/10.17487/rfc5155

[7] Topologically Integrated Geographic Encoding and Referencing (TIGER) Database, United States Census Bureau. https://www.census.gov/geo/maps-data/data/cbf/cbf_state.html

[8] Baugmgarten, H., Jung, H. and Mehlhorn, K. (1994) Dynamic Point Location in General Subdivisions. *Journal of Algorithms*, **17**, 342-380. https://doi.org/10.1006/jagm.1994.1040

[9] Dobkin, D. and Lipton, R.J. (1976) Multidimensional Searching Problems. *SIAM Journal on Computing*, **5**, 181-186. https://doi.org/10.1137/0205015

[10] Nekrich, Y. (2021) Dynamic Planar Point Location in Optimal Time. *Proceedings of the* 53*rd Annual ACM SIGACT Symposium on Theory of Computing*, Rome, 21-25 June 2021, 1003-1014. https://doi.org/10.1145/3406325.3451100

[11] Arya, S. and Mount, D.M. (2005) Computational Geometry: Proximity and Location. In: Mehta, D. and Sahni, S., Eds., *Handbook of Data Structures and Applications*, Chapman & Hall/CRC, Boca Raton, 22.

[12] Anagnostopoulos, A., Goodrich, M.T. and Tamassia, R. (2001) Persistent Authenticated Dictionaries and Their Applications. *Information Security Conference* (*ISC*), Vol. 2200, 379-393. https://doi.org/10.1007/3-540-45439-X_26

[13] Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A. and Stubblebine, S. (2001) A General Model for Authentic Data Publication. VC Davis Department of Computer Science Technical Report.

[14] Ramkumar, M. (2014) Symmetric Cryptographic Protocols. Springer, Berlin. https://doi.org/10.1007/978-3-319-07584-6

[15] Adhikari, N., Bushra, N. and Ramkumar, M. (2019) Complete Merkle Hash Trees for Large Dynamic Spatial Data. *The* 2019 *International Conference on Computational Science and Computational Intelligence* (*CSCI*'19), Las Vegas, 5-7 December 2019, 1318-1323. https://doi.org/10.1109/CSCI49370.2019.00246

[16] Chelladurai, U. and Pandian, S. (2021) HARE: A New Hash-Based Authenticated Reliable and Efficient Modified Merkle Tree Data Structure to Ensure Integrity of Data in the Healthcare Systems. *Journal of Ambient Intelligence and Humanized Computing*, 1-15. https://doi.org/10.1007/s12652-021-03085-0

[17] Goodrich, M.T., Tamassia, R. and Schwerin, A. (2001) Implementation of an Authenticated Dictionary with Skip Lists and Commutative Hashing. *DARPA Information Survivability Conference and Exposition*, Volume 2, 68-82.

[18] Ramkumar, M. (2018) Executing Large-Scale Processes in a Blockchain. *Journal of Capital Market Studies*, **2**, 106-120. https://doi.org/10.1108/JCMS-05-2018-0020

[19] Ramkumar, M. (2018) Scalable Computing in a Blockchain. *The* 2018 *IEEE Sarnoff Symposium*, Newark, 23-24 September 2018, 1-6. https://doi.org/10.1109/SARNOF.2018.8720499

[20] Dotan, M., *et al.* (2021) Survey on Blockchain Networking: Context, State-of-the-Art, Challenges. *ACM Computing Surveys* (*CSUR*), **54**, 1-34. https://doi.org/10.1145/3453161

[21] Ramkumar, M. (2019) A Blockchain Based Framework for Information System Integrity. *China Communications*, **16**, 1-17.

[22] Braden, B. (1986) The Surveyor's Area Formula. *The College Mathematics Journal*, **17**, 326-337. https://doi.org/10.1080/07468342.1986.11972974

[23] Merkle, R.C. (1987) A Digital Signature Based on a Conventional Encryption Function. *Conference on the Theory and Application of Cryptographic Techniques*, Santa Barbara, 16-20 August 1987, 369-378.

https://doi.org/10.1007/3-540-48184-2_32

[24] Adhikari, N., Bushra, N. and Ramkumar, M. (2017) Secure Queryable Dynamic Maps. *Enterprise Information Systems, and e-Government* (*EEE*'17), Las Vegas, 17-20 July 2017, pages.

[25] Reyes, I., Rollins, T., Mahnke, A. and Kadolph, C. (2014) Farm Mapping to Assist, Protect, and Prepare Emergency Responders: Farm MAPPER. *Journal of Agromedicine*, **19**, 232-233. https://doi.org/10.1080/1059924X.2014.888024