Efficient Tunable Location Data Private Information Retrieval with C2Sense

Nathan Ostrowski[1]
[1]*Duke University*

I. ABSTRACT

This paper presents C2SensePIR, a practical PIR scheme focused on serving location data with tunable privacy and efficiency. C2SensePIR considers an application where users seek path data from similar runners in their area. Our PIR scheme allows a user to leak a specific area of their query, linearly decreasing the size of the query vectors and the size of the database over which the server must perform multiplicative homomorphic computations with the area leaked. To do so, C2SensePIR orients the database around Hilbert values. This paper also focuses on presenting an ample background to PIR research and a comparison of cutting-edge work in the field.

II. BACKGROUND

A. Private Information Retrieval (PIR)

Fundamentally, the field of Private Information Retrieval (PIR) encompasses solutions to a basic problem: how can one interact with a server one doesn't trust? PIR considers a user with a specific query who wishes to submit its query to the server, compute some action over the data it stores, and receive information in return—but notably all without revealing information to an untrusted server. Broadly speaking, there are two main methods to do so: spreading computation out among non-colluding servers (Information Theoretic PIR, also known as IT-PIR) and using encryption methods to hide the user's intended query/result from the server (Computational PIR, also known as CPIR). We measure the usefulness of a PIR scheme by its computational and communicative costs. These include the relative time it takes a client to construct a query, the size of the query sent to the server, the relative time it takes the server to compute a result, and the size of the result sent back to the user. These costs may also include the communicative and computational costs of other operations not directly related to a query round, when a protocol uses stateful methods to persist storage on a client node and thus make an eventual query round more efficient.

B. Primitives

1. *k-anonymity*

k-anonymity describes a scheme where for some data, a malicious party should not be able to distinguish the identifiable parts of that data from among k other entries. Put in terms of data, for a tuple comprised of personal information and domain-specific information corresponding to a particular user (other than something akin to a uniqueID), there should be at least k other entries in that database with non-differentiable entries. As an example, consider a hospital database with patient entries, where the personal information is a dictionary of age, gender, weight, and blood type and the corresponding domain-specific data is a document hospital record, and the two are stored as a tuple in the database. A k-anonymity scheme would require at least k other tuples with the same personal information dictionary and different hospital record documents. The utility of a k-anonymity scheme is that someone seeking to de-anonymize a user by matching their physical attributes with a user on another database would not be able to identify any unique features of that user. The tuples, of course, do not have to correspond to actual users—but a high threshold of k may lead a system to populate its database with a large amount of dummy data, overwhelming the disk and causing considerable retrieval delays[1].

2. *Homomorphic Encryption*

Homomorphic encryption describes a set of functions one can use to modify encrypted data without ever decrypting its contents but still producing a correct (encrypted) result. A common analogy in the space is that of Alice and her jewelry shop. Alice imports jewels that need to be combined into necklaces, rings, and earrings, but happens to not trust the people assembling the products. Accordingly, Alice creates a locked box with two gloves leading into it, where an assembler may make the product without ever gaining access to it. Later, after the item is complete, Alice can come by and unlock the box. When applied to the problem of PIR, the assembler (the server) cannot see the jewelry (the user query). Put more formally, a Homomorphic Encryption algorithm is one that includes standard encryption methods of *KeyGen*, *Encrypt(Key)*, and *Decrypt(Key)* as well as a method *Evaluate(k, f, c1...cn)*, where k is some symmetric/public key, f is some arbitrary function over unencrypted inputs $m1...mn$, and $c1...cn$ are the output of *Encrypt(*$m1...mn$*)*. For the algorithm to be homomorphic, Evaluate ought to produce an encrypted c that reflects the output of function $f(m1...mn)$. Additionally, system architects also typically require that decrypting the final encrypted c be no more time intensive than decrypting c_i, and that c is the same size as $c1...cn$. The first classification of algorithms in this space com-

prises Fully Homomorphic Encryption algorithms (FHE) which typically operate in respect to two mathematical operations: addition and multiplication. FHE algorithms allow an oblivious worker to perform an unbounded number of operations on encrypted data. While ideal for security, they are widely impractical with current technology due to their extreme computational complexity. Thus, a new class of algorithms dubbed Somewhat/Leveled Homomorphic Encryption algorithms (SHE) were created to reduce computational complexity and thus increase practicality. SHE schemes differ from FHE schemes in that they cannot support an arbitrary number of mathematical operations on encrypted ciphertext. On the contrary, each operation creates noise on the encrypted result—and this noise may accumulate across SHE operations to render the final result unintelligible after decryption. The bulk of existing CPIR work covered in this paper (including XPIR, SealPIR, OnionPIR, and others) uses a BFV homomorphic scheme based on the Ring Learning with Errors (RLWE) cryptosystem. The BFV scheme operates over a plaintext ring and a ciphertext ring, while the homomorphism property allows one to map between the two rings while preserving operations. In the case of BFV, the plaintext polynomial ring is denoted by $P = R_t = \mathbb{Z}_t[x]/(x^n + 1)$, while the ciphertext polynomial ring can be denoted as $C = R_q \times R_q$ where $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. In these ring definitions, t and q are both in the span of Z, and because q is often much larger than t, C has a much higher cardinality than P, and thus one may map a plaintext p into many ciphertexts c, an essential property of an encryption method to prevent an adversary guessing and checking p. In this SHE scheme, multiplication is a significantly more expensive operation than addition, which may lead a system architect of the scheme to limit the bound of multiplication to include only low degree polynomials. For a crash course in the mathematics of BFV, BGV, and CKKS homomorphic encryption schemes, all of which are used in the existing work outlined in this paper, refer to Inferati's fantastically helpful guides[2].

C. Information Theoretic PIR (ITPIR)

As mentioned above, ITPIR schemes are multi-server schemes that operate under the assumption that a certain number of servers will not collude and provide each other with information that could be used to glean information about a user's query, result, or access patterns. ITPIR schemes are generally more efficient than their counterparts, CPIR schemes, but their onerous non-collusion assumption may disqualify their use for privacy-minded users. There are many ITPIR schemes, but perhaps the most noteworthy modern ITPIR scheme is Percy++, which has incredibly useful, tunable security properties. Namely, the Percy++ scheme is h-independent (the database is split between servers in some way so that no group of up to h servers can piece the database together), t-private (no group of up to t servers is privy to any information about the block the user requests), v-Byzantine-robust (as many as v servers can give incorrect replies, but the block is retrieved regardless), and k-out-of-m (out of m servers, the user only needs replies from k servers—the rest can be unresponsive). Put another way, the Percy++ scheme allows one to tune the amount of servers across whom (1) the database is divided, (2) collusion must be reached to produce meaningful findings, (3) a certain number may be bad actors, and (4) the final result is dependent on their participation. The Percy++ scheme's internals are notable for a number of reasons. First, the byzantine-fault-tolerance comes from an indirect application of Reed-Solomon error-correcting codes, where the client can first transform malicious errors from compromised servers into random errors, then correct these by interpreting the Reed-Solomon codewords. This transformation allows the scheme to remain intact even in the presence of a higher number of malicious servers, producing a correct result even in the face of significant sabotage[3][4].

D. Computational PIR (CPIR)

Contrasted to ITPIR schemes, CPIR schemes are single-server schemes that are protected by the strength of the encryption used as opposed to the distribution of the servers in communication (as in ITPIR). Therefore, CPIR schemes are useful because they avoid the significant trust assumption incurred by ITPIR that a certain number of servers must not collude. However, their cryptographic underpinnings tend to incur a signficant overhead. For context, xxx found that a modern CPIR scheme XPIR (described elsewhere in this paper) could process 22Gbps per second on their hardware, while the modern ITPIR scheme Percy++ could process 152Gbps on comparable hardware[5][6].

At a core level, CPIR techniques involve any obfuscation of requests and replies where the server cannot garner complete information about the information queried. Thus, there exist interesting approaches with weaker security (i.e. the server may garner some information about the user) but perhaps useful insights for how best to build a tunably-private solution. One such approach is to use a k-anonymity requirement (described elsewhere in this paper) with a Trusted Third Party (TTP) as a proxy to receive and anonymize (i.e. padding data or provide fake information such that the server cannot distinguish a user among k other users) before passing data on to the server[7]. Another approach is to use location obfuscation techniques to pass cloaking location data to the server in place of the user's exact location[8]. Still other methods pass entirely fake location data onto the server, continually receiving replies until the user has acquired each of the nearest neighbors and can construct a result on the client side [9]. In all of these methods, the user reserves some level of privacy from the server[10].

For stronger security, system architects typically use homomorphic encryption (described above). This allows them to construct CPIR protocols where the query may be encrypted, the user may perform some mathematical operations on that query over the database, and in doing so may produce an encrypted result that the user may decrypt (but the server may not). Many modern CPIR schemes (including Coeus[11], XPIR[6], OnionPIR[5], and SealPIR[12]) follow this basic pattern—albeit modified to include multiple rounds, batching, and other useful optimization techniques.

III. EXISTING WORK

A. CPIR Schemes

XPIR is a completely private, single-server CPIR scheme with a threat model excluding trusted hardware or trusted proxies. XPIR is built to be a malleable system, supporting varying encryption schemes (Paillier's and Ring-LWE methods including BFV). XPIR innovates by using a litany of mathematical tricks, including modifying the plaintext database to speed up the homomorphic addition and plaintext-ciphertext multiplication operations. As a result, compared to other modern CPIR methods, it is relatively fast for client-side and server-side computation, but has far greater communication complexity. More specifically, for a database with just over 1 million elements (where elements are of size 288 bytes) the XPIR scheme in a single round requires a query vector spanning 4,064KB and a response vector spanning 1,952KB[6][12][5].

SealPIR is another completely private, single-server CPIR scheme that builds on top of XPIR and has a similar threat model. SealPIR solely uses the Ring-LWE cryptosystem BFV (detailed by XPIR) but innovates by significantly bringing down communicative complexity and batching queries using probabilistic batch codes for improved server-side efficiency. While XPIR's query and response vectors require 4,046KB and 1,952KB respectively for a database with just over 1 million 288-byte elements, SealPIR's query and response vectors for a database of the same size require just 64KB and 256KB respectively. However, the total server computation costs are roughly tripled[12][5].

Coeus is a similar CPIR scheme focused around secure, private document retrieval. It implements a multi-round protocol to facilitate keyword searching over database contents and reduce communicativer complexity. The first round focuses on scoring a keyword query vector against elements in the database, the second around retrieving the metadata for the closest-matching documents, and the third around retrieving the entirety of the best-matching document. In its first (keyword scoring) round, Coeus uses term frequency-inverse document frequency (tf-idf) measures to judge relevance. On a high level, a tf-idf matrix assigns a weight to each phrase¡-¿document tuple, scoring how relevant the phrase is to that document. In a tf-idf matrix, the rows are the documents in the database and the columns are phrases. Thus, to score a given document relative to a user query, one can sum the tf-idf weights for all terms that are included in the user query. Because this is a matrix-vector multiplication operation, it is conducive to a homomorphic operation. The Coeus scheme is particularly useful when the size of individual elements in the database are large and one wishes to search through them. In its size optimizations, the Coeus scheme (particularly its third round) is comparable to a CPIR-exclusive and document-specific version of the Popcorn scheme, discussed elsewhere in this paper[11].

OnionPIR is another modern, completely private, single-server CPIR scheme built on top of both SealPIR and XPIR. OnionPIR innovates first by combining the Ring-LWE-based scheme BFV used by both SealPIR and XPIR with another Ring-LWE-based scheme RGSW for multiplicative homomorphic operations. XPIR and SealPIR use multiplicative homomorphic encryption operations between a plaintext database and a sequence of BFV ciphertext query vectors, where noise increases exponentially after each operation for the sequence of query vectors and the scheme must therefore split intermediate result ciphertexts between operations to avoid producing an undecipherable final result [1]. OnionPIR by contrast creates an additional query vector and composes its second and third queries out of RGSW ciphertexts. The scheme uses a sequence of external product operations between intermediate BFV and query RGSW ciphertexts, where noise only increases additively. Therefore, OnionPIR avoids the intermediate plaintext and final result ciphertext duplication incurred by XPIR and SealPIR. Similar to SealPIR, it also introduces a batching system for increased efficiency. Finally, OnionPIR innovates by using Chen et al.'s query compression algorithm, where the client packs multiple independent bits into a single ciphertext, which the server obliviously un-

[1] For reference, in the BFV scheme, when an mxn plaintext matrix is multiplied by a $m \times 1$ BFV ciphertext vector, the operation produces a $1 \times n$ BFV ciphertext vector. A scheme attempting to single out an element of the original plaintext matrix using BFV SHE operations must use a pair of query vectors, where one selects a row of the plaintext matrix and another selects a column. After the initial BFV first query vector and plaintext database multiplicative operation produces an intermediate $1 \times n$ BFV row vector, that intermediate $1 \times n$ BFV vector must be multiplied homomorphically with the second $1 \times n$ BFV query vector to produce a resulting BFV vector with the desired result. Because the multiplication between BFV vectors adds so much noise, schemes must separate out the intermediate $1 \times n$ BFV row vector into a $n \times n$ plaintext matrix, then perform a multiplicative operation between the intermediate $n \times n$ plaintext matrix and the $1 \times n$ BFV for vector to produce a $m \times 1$ BFV result vector, where each element of the vector is the duplicated retrieved element. Obviously, this scheme incurs quite a lot of redundancy, from splitting of the intermediate vector into a plaintext matrix to producing an m-duplicated final result vector.

packs into single-bit encryptions. This vastly decreases the request size, but increases noise (per bit, in a multiplicative manner) when more independent bits are packed into a single ciphertext. To thus limit the number of bits packed—reducing the query vector size while maintaining a manageable level of noise growth—the authors represent the database as a high-dimensional hypercube[5].

B. Combined ITPIR/CPIR Schemes

Popcorn considers the use case of a streaming service in PIR, where users seek to access media while not revealing information about the media they access to the service. Popcorn combines CPIR and ITPIR for a hybrid scheme. In its ITPIR component, Popcorn relies upon the trust assumption that individual servers may be malicious but they may not collude. As such, Popcorn's scheme centers around a tree of content distributor nodes, where the root node creates an encrypted version of the media library using per-object keys, then passes this encrypted library down to secondary nodes in separate CDNs. The root node maintains a key server, while each child maintains a corresponding object server with the entire database. In its CPIR component, the user communicates with the root node key server to obtain the keys necessary to decode the encrypted chunks served by ITPIR between the user and the child object libraries. These steps can be taken in parallel, with the user applying a decode algorithm to the encrypted ITPIR-received chunks using the CPIR-received key. Popcorn uses the CPIR implementation of a related scheme called XPIR, an algorithm (described elsewhere in this paper) based on the Ring-LWE class of cryptographic operations which performs partially homomorphic encryption on BFV vectors subject to a noise constraint. Popcorn uses the ITPIR implementation of Percy++, detailed above. To reduce the onerousness of disk operations, Popcorn also pipelines query processing in its ITPIR component, eliminating the I/O bottleneck for a database with large elements and thus signficantly speeding up the scheme[13].

C. Trusted Hardware PIR Schemes

Trusted Hardware PIR techiniques are typically used where the contents of a database are private (i.e. encrypted) and their storage is outsourced by a user to the server, or where user-private and public entries co-exist on a database. Asonov was the first noted researcher to propose a PIR scheme using trusted hardware, with the advantage of vastly improved O(1) communication cost in client-server communication, contrasted to far more costly ITPIR and CPIR methods. Asonov's scheme using a secure hardware module still requires the server to scan the entirety of the database when processing each request, however, so its retrieval computational complexity server-side remains O(n), where n is the database length[14].

Iliev and Smith from Dartmouth consider an Oblivious RAM (ORAM) model, where the database may contain private entries, public entries, or both, and the goal is to hide their access patterns from a curious server. ORAM schemes often use PIR, and there are interesting modern examples of this dual application[15]. In Iliev and Smith's scheme, a trusted hardware module encrypts all records in the database, then securely shuffles all encrypted entries by a permutation number pi, but does so using an algorithm (Beneš permutation networks, with O(N log N) complexity) where the pattern observed by the untrusted host is independent of pi. After this encrypting/shuffling, the data retrieval portion begins. Because the goal is to keep the untrusted server from learning about data access patterns to the records, for each record fetch in the database, the secure hardware fetches the intended encrypted record as well as all previously queried records. This recursive operation ensures that the intended record will be in the scope of those retrieved, but the server is none-the-wiser about the identity of the record. Here, the computational complexity is O(i), where i is some maximum depth of recursive record fetching in a given data retrieval session, before the trusted hardware module must re-shuffle the encrypted database.[16]

Snoopy is a modern scheme that leverages trusted hardware to perform load balancing in addition to standard encrypting and shuffling. Like Iliev and Smith, this scheme can accomodate any privacy of database entries, with the key difference of being custom-built for a high-throughput system and achieving comparatively terrific efficiency by dividing traditional ORAM tasks[17].

Location-based applications are a particulary exciting area of research full of recent innovation in PIR schemes[18][19]. Fung et al. propose another trusted hardware PIR scheme focused around location-based applications (i.e. Google Maps, Yelp, etc.). Fung et al. incorporate a clever scheme for distributing location information for more efficient PIR, encoding relative proximity between locations using an imposed Hilbert grid, and storing the cells in the order of their Hilbert values. Fung et al. also use a triple-database partition scheme to split up the data corresponding to points of interest. On the first database, they store Hilbert-ordered point-of-interest counts, on the second, longitudes and latitudes of points-of-interest, and on the third, extra information about each point-of-interest. A client issues a sequence of k-nearest-neighbor queries to the trusted hardware running on the server, with the server providing increasingly focused and complex information. Fung et al. also protect against information leakage using fixed-size queries across the three communication rounds. Their scheme offers interesting insight into the usefulness of multiple databases in a location-focused protocol to facilitate a multi-round query with varying levels of information depth to speed up server-side computation. However, their use of server-side trusted hardware to achieve this

goal represents a significant limitation[10].

IV. PROBLEM STATEMENT

This paper considers the specific use-case of a sensing application serving information about the running habits and routes of similar users in a given area. A user logging into the application provides the application with their age, weight, BMI, and continual access to their location. The application then continually monitors the habits of the user to determine their top activity types (i.e. long-distance-runner, city-runner, etc.), though the process for determining this activity type is not important for the context of this paper. The user at some point wants to know the common running routes in a particular *area* for users with a similar age, weight, BMI, and activity types. A fully private PIR scheme would ensure that no information about the *area* of a search or the similar attributes of the user matched on are leaked. However, such schemes often incur significant overhead as a consequence of needing to perform CPIR operations over the entirety of the database (see footnote 1 and the Existing Work discussion of XPIR, SealPIR, and OnionPIR for a more thorough discussion of CPIR complexity and tradeoffs). Therefore, this scheme responds to the question: can we leak some controlled information to the server in return for a much faster response?

V. THREAT MODEL

In this scheme, we consider a database whose contents are public (as in many consumer-facing services, such as Google Maps, Yelp, Wikipedia, etc.). We also consider the presence of an honest but curious adversary (i.e. the adversary does not have unlimited time and resources, and will be sufficiently limited by the onerousness of decrypting any particular communication between the client and server). Our trusted computing base includes client-side secure hardware modules and the codebase used for homomorphic encryption, query generation, and result processing operations. We consider speculative execution attacks on this client-side trusted hardware out of scope. Data integrity and denial of service are similarly orthogonal problems, and both out of scope.

VI. PROTOCOL

The key goal of PIR on C2Sense is to allow the user to leak some discrete information to the server in exchange for a faster result. Many PIR schemes suffer from impracticality as a result of striving for complete privacy across the protocol[5][6][11][12]. As mentioned in II, the notion of a faster result can be broken down into the client-side computation needed to generate a request vector, the communication complexity of the request vector, the computation time of homomorphic operations undertaken by the server, the communication complexity of the return vector, and the time needed by the client to process the return vector. The PIR protocol of C2Sense optimizes for lowest communication complexity while maintaining a comparable computation time for server-side homomorphic operations. C2SensePIR is most similar to the mechanics of OnionPIR[5], with a similar cryptographic basis for query generation and homomorphic operations steps, but with modified query and database structures for flexible privacy/efficiency.

In the particular use case outlined in IV, the user seeks route location data for runners similar to them, whose route points are stored in buckets—arrays holding common route points for runners with particular attributes. The user knows and stores a mapping of this public database.

The key insight of the C2SensePIR protocol is to exploit the properties of a Hilbert Curve to allow the user to leak some finite area L in order to reduce the area of the database on which to perform homomorphic operations (and thus—to reduce the query size, result size, server-side computation complexity, and server-side intermediate matrix size).

A. Hilbert Curve

A Hilbert curve is a continuous, space-filling data structure whose length grows exponentially, where each individual curve is contained in a square with an area of 1. In 2 dimensions, we may impose a grid over a square space, and have points on this grid map to Hilbert values. Fig. 1 shows the progression of a Hilbert curve from its first to third orders.

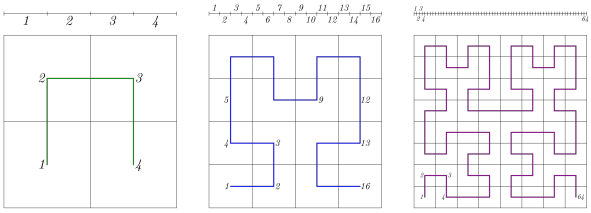

FIG. 1. First to third orders of the Hilbert curve visualized. Source: Wikipedia.

The key fact about the Hilbert curve that allows one to use it to build a flexibly private PIR system is the fact that Hilbert values for subspaces on a further order of the Hilbert curve *maintain the same leading bits*. This is demonstrated in Fig. 2 for an arbitrary subspace $H(N)$ with binary Hilbert value h, where each of the 4 subspaces of $H(N)$ can be uniquely represented by concatenating h with just two bits.

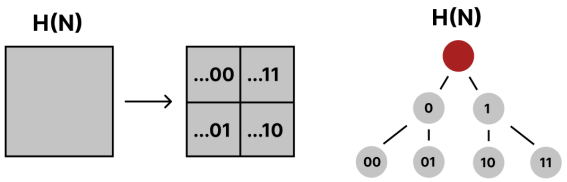

FIG. 2. Left: Representation of the leading-bit-preserving property of the Hilbert curve, where any subcurve can be expressed by appending bits to its parent. Right: Breakdown of an efficient binary tree data structure to store and retrieve location data according to their Hilbert values.

B. Database construction

At the time of information processing, after a sensing "round" (when all participating users have submitted their data) the server takes all of the raw location data provided by users (this collection is an orthogonal problem, described in VII), then imposes a Hilbert curve of order 6 (to produce a 64-bit Hilbert value, though this should be explored in VII) categorizes the data into a binary tree data structure (outlined in Fig. 2) by performing the following recursive algorithm:

- Takes as its inputs a representation of the bits left to navigate h, the current node n, and a tuple of the location point with the bucket (i.e. user type) it corresponds to $p : b_u$.
    1. If there are no bits left to navigate, stop (this indicates the leaf).
        (a) If the array b_u does not already exist, create it, and pad it with 3 junk entries (all bits 0)
        (b) Else, if b_u does exist, check for junk entries
            i. If a junk entry exists, dump from array
            ii. Else, dump the p with the lowest c value corresponding to the count of users contributing to this point
        (c) Append p to b_u
    2. Else, consider the t as the n_{th} bit of h:
        (a) Create and link the child node corresponding to t if it does not already exist
        (b) Recurse, where $h = h_1...h_{n-1}$, $n = t$, and $p : b_u$ is preserved.

This algorithm produces the database structure illustrated by Fig. 3. The padding of buckets placed in the database is presently space-expensive to protect the threat model requirement that the server learn nothing about the number of points (i.e. the popularity) of the very specific location/user group queried. Presently, due to this junk data, the response size will remain fixed. Future work should consider whether a smaller-ordered Hilbert curve ought to be used when generating Hilbert values for contributed location points in order to increase the incidence rate of multiple points in a single bucket and decrease the likelihood that a bucket will contain padded junk data. However, any solution that increases this incidence rate may incur the opposite problem that non-junk entries are often booted from a bucket set.

To send a query—consistent with the query-size-minimizing approach explored by OnionPIR[5] whose intricacies are explored in III—the client application first retrieves information about the layout of the database through a communication round with the server, then produces a sequence of ciphertext query vectors encoding information about the desired bucket (i.e. for a user profile similar to theirs). Here, the client and server represent the database as an $n \times n$ (where n is an even number) square matrix of binary blocks of bucket size. The 1st $n \times 1$ *BFV* ciphertext query vector encodes a 1 at the index of the desired row of the database queried, while the 2nd and 3rd $1 \times \frac{n}{2}$ *RGSW* ciphertext query vectors encode a 1 at the further partitions of the plaintext database corresponding to the bucket they would like to access. The server processing the computation takes the 1st $n \times 1$ *BFV* ciphertext query vector and performs a BFV ciphertext-plaintext multiplication with the plaintext database. This produces a *BFV* ciphertext intermediate product, encoding a row in the database. The server then takes an external product of this *BFV* ciphertext intermediate product with the 2nd query vector (a $1 \times \frac{n}{2}$ *RGSW* ciphertext vector) to produce another *BFV* intermediate result corresponding to a set of $\frac{n}{2}$ entries in the row. Taking a 3rd external product with this 2nd *BFV* intermediate result and the 3rd query vector (a $\frac{n}{2} \times 1$ *RGSW* ciphertext vector), the server finally produces a *BFV* ciphertext object encoding the bucket the user wishes to retrieve, then sends it back as a result object to the user. The client device then decrypts this object using the private key corresponding to the public key it used to create the ciphertext query vectors.

Where C2SensePIR radically departs from the CPIR scheme presented by OnionPIR is that addition to the sequence of query vectors, the user *may also send the server some L such that L is less than the Hilbert value for the desired bucket.* In doing so, the user may instead send query vectors of length g that correspond to just the Hilbert values contained as a subset of the leading bits provided by L. Instead of an expensive homomorphic multiplication computation over the entire database, the server may now perform this operation *only over the range of the database whose leadings bits match L*—the child nodes of the binary tree whose root is given by recursing with L. A representation of the subset specified by L is given in Fig. 3.

Furthermore, we can increase the accuracy of locality measured by the Hilbert curve imposition by splitting the representation of Earth into the 6 faces of a cube. As key background, consider Google's s2 library, (taken

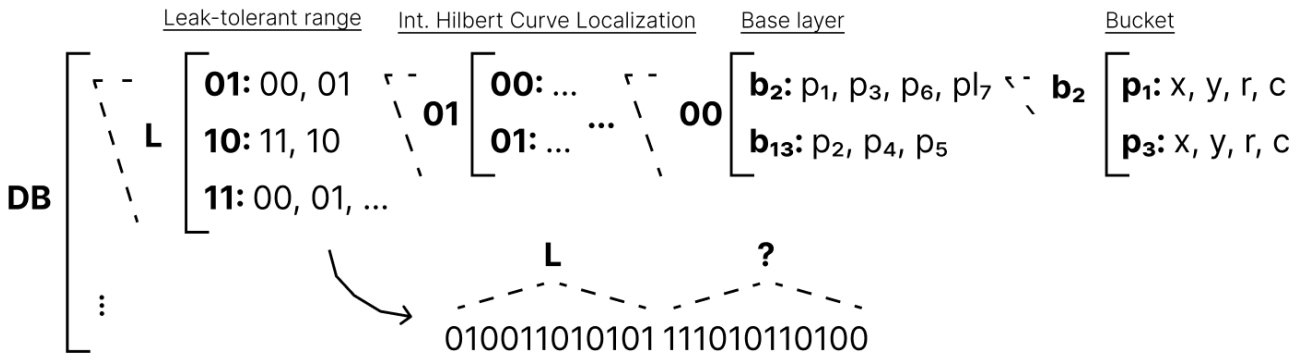

FIG. 3. Representation of the C2SensePIR Database Structure. The user has specified some L Hilbert value, so the server need only consider elements whose Hilbert values match the leading bits provided by L (Also interpreted as the child nodes of the node found by recursing with L in Fig. 2) as the range of the database over which to perform homomorphic operations.

from the name for a unit sphere) which aims to spatially index then retrieve location data efficiently in the face of growing location data sets. First, s2 contrasts from traditional Mercator projection mappings of location data by representing Earth not as a flat surface, but as the 6 faces of a cube, with further non-linear transformations applied to reduce deformations. This 6-face projection (where each face is a quadtree) drastically reduces distortion and improves real locality of data over simple 2d projections [20]. This projection (and its 64-bit Cell ID representation, helpfully diagrammed here and further described here) represent an intuitive way to address location information which one can use to split up among 6 binary trees with maximum heights of 30, though further work ought to be done to determine the degree of specificity needed by this particular problem statement.

VII. CONCLUSION

C2SensePIR is a practical, tunable PIR scheme that allows a user to leak a specific area of their query to linearly reduce the size of the query vectors communicated to the server and linearly reduce the computational complexity of the server's homomorphic operations—thus serving the result faster. C2SensePIR achieves this goal by imposing a Hilbert curve over aggregated location data, orienting the database around the produced Hilbert values associated with each point. C2Sense uses a sequence of query vectors sent to the server along with this leakage area L, and receives an encrypted bucket object in return, where the bucket corresponds to an array of path locations for similar users.

Future work ought to explore optimal methods of batching (building off of similar CPIR schemes including SealPIR and OnionPIR) as well as optimal techniques for Stateful PIR (building off of OnionPIR). An important orthogonal problem is anonymous data collection and aggregation, which precedes the database construction steps described in VI. More research should also be done on efficient schemes for retrieving multiple elements, (that do not simply run as many single-retrieval protocols), known as multi-retrieval PIR techniques, including the work of Angel et al. and Henry et al. Future work may also consider the implications of including the Ring-LWE-based scheme CKKS. Orthogonal work may also consider narrowing the trusted computing base on the client side (including treating the client OS as untrusted and considering side-channel mitigations such as retpolines for code running on client-side trusted hardware). The problem of private information retrieval is pertinent to everyone who wishes to keep their privacy in their own hands, and how a user ought to tune a system depends on their privacy comfort level. Thus, future research ought to focus on how best to present users with clear privacy-efficiency trade-offs.

[1] L. Sweeney, "K-anonymity: A model for protecting privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, p. 557–570, oct 2002.

[2] I. Inc, "Introduction to the bfv encryption scheme," Feb. 2021.

[3] C. Devet, I. Goldberg, and N. Heninger, "Optimally robust private information retrieval," in *21st USENIX Security Symposium (USENIX Security 12)*, (Bellevue, WA), pp. 269–283, USENIX Association, Aug. 2012.

[4] S. M. Hafiz, *Private Information Retrieval in Practice*. PhD thesis, 2021. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2021-05-11.

[5] M. H. Mughees, H. Chen, and L. Ren, "Onionpir: Response efficient single-server pir," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Com-*

munications Security, CCS '21, (New York, NY, USA), p. 2292–2306, Association for Computing Machinery, 2021.

[6] C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian, "Xpir : Private information retrieval for everyone," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 2, pp. 155–174, 2015.

[7] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias, "Preventing location-based identity inference in anonymous spatial queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 12, pp. 1719–1733, 2007.

[8] H. Kido, Y. Yanagisawa, and T. Satoh, "An anonymous communication technique using dummies for location-based services," vol. 88-97, pp. 88– 97, 08 2005.

[9] M. Yiu, C. Jensen, X. Huang, and H. Lu, "Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services," pp. 366–375, 05 2008.

[10] E. Fung, G. Kellaris, and D. Papadias, "Combining differential privacy and pir for efficient strong location privacy," pp. 295–312, 08 2015.

[11] I. Ahmad, L. Sarker, D. Agrawal, A. El Abbadi, and T. Gupta, "Coeus: A system for oblivious document ranking and retrieval," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, (New York, NY, USA), p. 672–690, Association for Computing Machinery, 2021.

[12] S. G. Angel, H. Chen, K. Laine, and S. T. V. Setty, "Pir with compressed queries and amortized query processing," *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 962–979, 2018.

[13] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish, "Scalable and private media consumption with popcorn," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, (Santa Clara, CA), pp. 91–107, USENIX Association, Mar. 2016.

[14] P. Williams and R. Sion, "Usable pir," in *NDSS*, 2008.

[15] Z. Zhang, K. Wang, W. Lin, A. W.-C. Fu, and R. C.-W. Wong, "Practical access pattern privacy by combining pir and oblivious shuffle," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, (New York, NY, USA), p. 1331–1340, Association for Computing Machinery, 2019.

[16] A. Iliev and S. Smith, "Protecting client privacy with trusted computing at the server," *IEEE Security Privacy*, vol. 3, no. 2, pp. 20–28, 2005.

[17] E. Dauterman, V. Fang, I. Demertzis, N. Crooks, and R. A. Popa, "Snoopy: Surpassing the scalability bottleneck of oblivious storage," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, (New York, NY, USA), p. 655–671, Association for Computing Machinery, 2021.

[18] F. Ye and S. E. Rouayheb, "Intermittent private information retrieval with application to location privacy," 2021.

[19] J. Zhang, C. Li, and B. Wang, "A performance tunable cpir-based privacy protection method for location based service," *Information Sciences*, vol. 589, pp. 440–458, 2022.

[20] S2Geometry, "S2 cells: Developer guide," Apr. 2022.