Enhancing the Performance of Spatial Queries on Encrypted Data Through Graph Embedding

Sina Shaham, Gabriel Ghinita, Cyrus Shahabi

Enhancing the Performance of Spatial Queries on Encrypted Data through Graph Embedding

Sina Shaham[1], Gabriel Ghinita[2], and Cyrus Shahabi[1]

[1]University of Southern California, [2]UMass Boston

Abstract. Most online mobile services make use of location data to improve customer experience. Mobile users can locate points of interest near them, or can receive recommendations tailored to their whereabouts. However, serious privacy concerns arise when location data is revealed in clear to service providers. Several solutions employ *Searchable Encryption (SE)* to evaluate spatial predicates directly on location ciphertexts. While doing so preserves privacy, the performance overhead incurred is high. We focus on a prominent SE technique in the public-key setting – *Hidden Vector Encryption (HVE)*, and propose a graph embedding technique to encode location data in a way that significantly boosts the performance of processing on ciphertexts. We show that finding the optimal encoding is NP-hard, and provide several heuristics that are fast and obtain significant performance gains. Our extensive experimental evaluation shows that our solutions can improve computational overhead by a factor of two compared to the baseline.

Keywords: Hidden Vector Encryption · Graph Embedding.

1 Introduction

Location data play an important part in offering customized services to mobile users. Whether they are used to find nearby points of interest, to offer location-based recommendations, or to locate friends situated in proximity to each other, location data significantly enrich the type of interactions between users and their favorite services. However, current service providers collect location data in clear, and often share it with third parties, compromising users' privacy. Movement data can disclose sensitive details about an individual's health status, political orientation, alternative lifestyles, etc. Hence, it is important to support such location-based interactions while protecting privacy.

Our focus is on *secure alert zones*, a type of location-based service where users report their locations in encrypted form to a service provider, and then they receive alerts when an event of interest occurs in their proximity. This operation is very relevant to contact tracing, which is proving to be essential in controlling pandemics, e.g., COVID-19. It is important to determine if a mobile user came in close proximity to an infected person, or to a surface that has been exposed to the virus, but at the same time one must prevent against intrusive surveillance of the population. More applications of alert zones include notifications in the case of other natural disasters, or even commercial applications (e.g., notifying mobile users of nearby sales events).

Searchable Encryption (SE) [3,14,20] is very suitable for implementing secure alert zones. Users encrypt their location before sending it to the service provider, and a special kind of encryption is used, which allows the evaluation of predicates directly on ciphertexts. However, the underlying encryption functions are not specifically designed for geospatial queries, but for arbitrary keyword or range queries. As a result, a data mapping step is typically performed to transform spatial queries to the primitive operations supported on ciphertexts. Due to this translation, the performance overhead can be significant. Some solutions use *Symmetric Searchable Encryption (SSE)* [6,14,20], where a trusted entity knows the secret key of the transformation, and collects the location of all users before encrypting them and sending the ciphertext to the service provider. While the performance of SSE can be quite good, the system model that requires mobile users to share their cleartext locations with a trusted service is not adequate from a privacy perspective, since it still incurs a significant amount of disclosure.

To address the shortcomings of SSE models, the work in [3] introduced the novel concept of *Hidden Vector Encryption (HVE)*, which is an *asymmetric* type of encryption that allows direct evaluation of predicates on top of ciphertext. Each user encrypts her own location using the *public* key of the transformation, and no trusted component that accesses locations in clear is required. This approach has been considered in the location context in [10], [16], with encouraging results. However, the performance overhead of HVE in the spatial domain remains high. Motivated by this fact, we study techniques to reduce the computational overhead of HVE. Specifically, we derive special types of spatial data mapping using graph embeddings, which allow us to express spatial queries with predicates that are less computationally-intensive to evaluate.

In existing HVE work for geospatial data [10], [16], the data domain is partitioned into a hierarchical data structure, and each node in this structure is assigned a binary string identifier. The binary representation of each node plays an important part in the query encoding, and it influences the amount of computation that needs to be executed when evaluating predicates on ciphertexts. However, the impact of the specific encoding is not evaluated in-depth. Our approach embeds the geospatial data domain to a high-dimensional hypercube, and then it applies graph embedding [4] techniques that directly target the reduction of computation overhead in the predicate evaluation step.

Our specific contributions are:

- We introduce a novel transformation of the spatial data domain based on graph embedding that is able to model accurately the performance overhead incurred when running HVE queries for spatial predicates;
- We transform the problem of minimizing HVE computation to a graph problem, and show that the optimal solution is NP-hard;
- We devise several heuristics that can solve the problem efficiently in the embedded space, while still reducing significantly the computational overhead;
- We perform an extensive experimental evaluation which shows that the proposed approaches are able to halve the performance overhead incurred by HVE when processing spatial queries.

The rest of the paper is organized as follows: Section 2 introduces necessary background on the system model (an HVE primer is given in Appendix A). Section 3 provides the details of the proposed graph embedding transformation. Section 4 formulates the problem of minimizing the amount of required computation with HVE in the embedded space, and proves that the optimal solution is NP-hard. Section 5 introduces several heuristic algorithms that solve the problem efficiently. Section 6 evaluates thoroughly the proposed approach on real-life datasets. We survey related work in Section 7 and conclude in Section 8.

2 Background

2.1 System Model

Consider a [0,1]×[0,1] spatial data domain divided into n non-overlapping partitions, denoted as $\mathcal{V} = \{v_1, , v_2, ..., v_n\}$. We use the term *cell* to refer to partitions; however, a cell can have an arbitrary size and shape. An example of such a partitioning is provided in Fig. 2a. The system architecture of location-based alert system is represented in Fig. 1, and consists of three types of entities:

1. **Mobile Users** subscribe to the alert system and periodically submit encrypted location updates.
2. The **Trusted Authority (TA)** is a trusted entity that decides which are the alert zones, and creates for each zone a search *token* that allows to check privately if a user location falls within the alert zone or not.
3. The **Server (S)** is the provider of the alert service. It receives encrypted updates from users and search tokens from TA, and performs the predicate evaluation to decide whether encrypted location C_i of user i falls within alert zone j represented by token TK_j. If the predicate holds, the server learns message M_i encrypted by the user, otherwise it learns nothing.

The system supports location-based *alerts*, with the following semantics: a *Trusted Authority (TA)* designates a subset of cells as an *alert zone*, and all the users enclosed by those cells must be notified. The TA can be, for instance, the Center for Disease Control (CDC), who is monitoring cases of a pandemic, and wishes to notify users who may have been affected; or, the TA can be some commercial entity that the users subscribe to, and who notifies users when a sales event occurs at selected locations.

The *privacy requirement* of the system dictates that the server must not learn any information about the user locations, other than what can be derived from the match outcome, i.e., whether the user is in a particular alert zone or not. In case of a successful match, the server S learns that user u is enclosed by zone z. In case of a non-match, the server S learns only that the user is outside the zone z, but no additional location information. Note that, this model is applicable to many real-life scenarios. For instance, users wish to keep their location private most of the time, but they want to be immediately notified if they enter a zone where their personal safety may be threatened. Furthermore, the extent of alert zones is typically small compared to the entire data domain, so the fact that S learns that u is *not* within the set of alert zones does not disclose significant

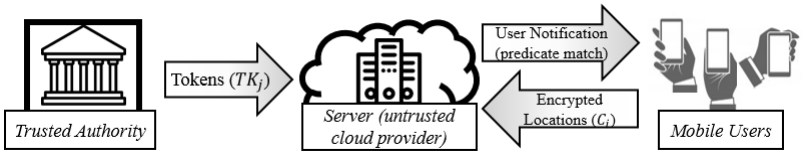
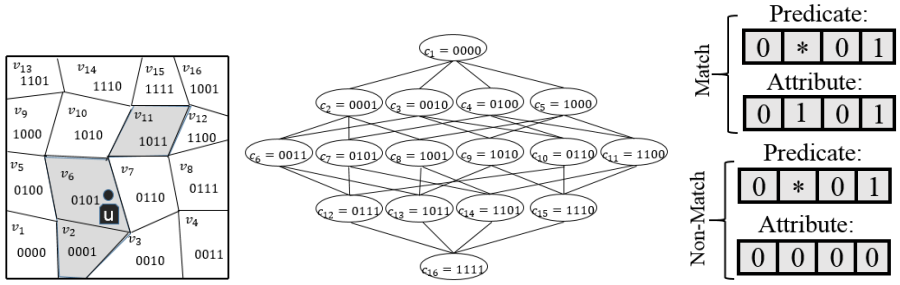


Fig. 1: Location-based alert system.



(a) Sample grid. (b) Graph $G_1(\mathcal{C}, \mathcal{E}_1)$. (c) HVE evaluation.

Fig. 2: An example of embedding graphs generated based on a sample grid.

information about u's location. The TA can be an organization such as a city's public emergency department, which is trusted not to compromise user privacy, but at the same time does not have the infrastructure to monitor a large user population, and outsources the service to a cloud provider.

2.2 Problem Statement

Prior work assumed that all cells are equally likely be in an alert zone. However, that is not the case in practice. Some parts of the data domain (e.g., denser areas of a city) are more likely to become alert zones. The cost of encrypted alert zone enclosure evaluation is given by the number of operations required to apply HVE matching at the service provider. As we discuss in our HVE primer in Appendix A, the evaluation cost is directly proportional to the number of non-star bits in the tokens. Armed with knowledge about the likelihood of cells to be part of an alert zone, one can create superior encodings that reduce processing overhead.

Our goal is to find an enhanced encoding that reduces non-star bits for a given set of alert zone tokens. Denote by $p(v_i)$ the probability of cell v_i being part of an alert zone. The *mutual* probability of multiple cells indicates how likely they are to be part of the *same* alert zone. Given individual cell probabilities, the mutual probability of a set of i cells $\mathcal{L} = \{v'_1, , v'_2, ..., v'_i\}$ is calculated as:

$$p(\mathcal{L}) = \prod_{j=1}^{i} p(v'_j). \tag{1}$$

The problem we study is formally presented as follows:

Problem 1. Find an encoding of the grid that reduces the number of non-star bits in the tokens generated from alert zone cells.

3 Location Domain Mapping through Graph Embedding

Our approach minimizes the number of non-star bits in alert zone tokens by modeling the data domain partitioning as an embedding problem of a k-cube into a complete graph. We denote a k-cube as $G_1(\mathcal{C}, \mathcal{E}_1)$, where $\mathcal{C} = \{c_1, , c_2, ..., c_n\}$ and $c_i = \{0,1\}^k$. Fig. 2b illustrates a k-cube generated based on the sample partitioning in Fig 2a. In G_1, two nodes c_i and c_j are connected if their *Hamming distance* is equal to one. We refer to such a bit as *Hamming bit*.

Definition 1 *(Hamming Distance and Bits). The Hamming distance between two indices c_i and c_j in $G_1(\mathcal{C}, \mathcal{E}_1)$ is the minimum number of substitutions required to transform c_i to c_j, denoted by the function $d_h(.)$. We refer to the bits that need to be transformed as the Hamming bits of indices.*

Example 1 *The Hamming distance between indices $c_1 = 0100$ and $c_2 = 0010$ is two ($d_h(c_i, c_j) = 2$), and the Hamming bits are the second and third most significant bits of the indices.*

The second graph required to formulate the problem of minimizing the number of non-stars is a complete graph generated by all cells in the partitioning, denoted by $G_2(\mathcal{V}, \mathcal{E}_2)$. The set $\mathcal{V}$ represents the nodes corresponding to cells, and an undirected edge connects every two nodes in G_2.

Note that, every token (including those containing stars), can be related to several cycles on the k-cube. For example, token 00** represents four indices $0000, 0001, 0010, 0011$, which correspond to cycles (c_1, c_2, c_6, c_3) and (c_1, c_3, c_6, c_2) on the k-cube in Fig. 2b. Unfortunately, there is no one-to-one correspondence between the tokens and the cycles. In particular, for a larger number of stars, there exist several cycles representing the same token. To generate a one-to-one correspondence, we incorporate *Binary-Reflected Gray (BRG)* encoding on the k-cube to create unique cycles corresponding to tokens.

Definition 2 *(BRG path on k-cube). A BRG path between two nodes with non-zero Hamming distance is defined as the path on the k-cube going from one node to another based on BRG coding on Hamming bits.*

As an example, the Hamming bits between 0001 and 1000 are the least and most significant bits, and the BRG path connecting them on the k-cube in Fig.2b includes indices 0001, 1001, and 1000 in the given order. One can see that as the BRG codes are unique, the BRG path between two indices on the k-cube is also unique. This characteristic of BRG paths is formulated in Lemma 1.

Lemma 1. *A BRG path between two nodes on a k-cube is unique.*

Proof. The uniqueness of the path between two nodes on the k-cube follows from the uniqueness of BRG code, as only one such a path can be constructed.

Definition 3 *(Complete x-bit BRG cycle). Given a k-cube, a complete x-bit BRG cycle is a cyclic BRG path with the length of 2^x, in which only x bits are affected. We denote the set of all possible complete x-bit BRG cycles by $\mathcal{L}_x = \{\bigcup l_i\}$.*

Example 2 *In Fig. 2b, token *0** entails eight indices 0000, 0001, 0011, 0010, 1010, 1011, 1001, 1000. This token maps uniquely to the complete 3-bit BRG cycle on the 4-cube with nodes* (c_1, c_2, c_6, c_3, c_9, c_{13}, c_8, c_5) *and start point* c_1.

We can uniquely associate a token to a cycle on the k-cube. Consider a token with k bits and x stars. This token is mapped to a complete x-bit BRG cycle on the k-cube, starting from a node in which all the star bits are set to zero. Such a cycle is unique and has a length of 2^x. Based on this mapping, every token is associated with a unique cycle on the k-cube, and every complete x-bit BRG cycle is mapped to a unique token with x-stars. Therefore, there is a one-to-one correspondence between tokens and complete BRG cycles. The formulation of Problem 1 based on graph embedding can be written as follows:

Problem 2. Given two graphs $G_1(\mathcal{C}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$, find a mapping function $\mathcal{F} : G_1 \rightarrow G_2$ with the objective to

$$Maximize\{\sum_{i=1}^{k} p(\mathcal{L}_i)\}. \tag{2}$$

3.1 Gray Optimizer (GO)

The problem of embedding a complete graph within a minimized size k-cube has been shown to be NP-hard [4]. We develop an heuristic algorithm called *Gray Optimizer* that solves Problem 2. Specifically, we determine an optimal embedding with respect to a given node in $\mathcal{G}_2$ with complexity of $\mathcal{O}(n(\log_2 n)^4)$. Consider an initial node of the complete graph $v_r \in \mathcal{V}$, and without loss of generality assume that it is assigned to index[1] c_1. The optimization problem can be formulated as follows.

Problem 3. Given two graphs $G_1(\mathcal{C}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$, and the node $v_r \in \mathcal{V}$ assigned to index c_1, find a mapping function $\mathcal{F} : G_1 \rightarrow G_2$ that

$$Maximize\{\sum_{i=1}^{k} p(\mathcal{L}_i|v_r)\}. \tag{3}$$

Problem 2 requires an assignment of vertices in G_2 to the nodes of G_1 such that the probability of complete BRG cycles is maximized; whereas Problem 3 seeks to maximize the probability of cycles with respect to a particular node, in this case v_r, which is assigned to the index c_1. A reasonable candidate for assignment to c_1 is the cell with the highest probability, as it is most likely to be part of an alert zone. To solve this problem, we propose the heuristic in Algorithm 1.The input of the algorithm is the root index $c_1 \in G_1$, the root node $v_r \in G_2$ (also called seed) and the graphs G_1 and G_2.

Denote by $\mathcal{D}_{i|c_1}$ the set of nodes on $\mathcal{C}$ that have a Hamming distance of i from c_1. Note that $\mathcal{D}_{i|c_1}$ includes $\binom{k}{i}$ nodes, each one having a Hamming distance of i from c_1. The overall assignment structure is as follows: first, Algorithm 1 assigns the remaining nodes of $\mathcal{V}$ of the graph $\mathcal{G}_2$ to nodes in $\mathcal{D}_{1|c_1}$. After assignment of all nodes in $\mathcal{D}_{1|c_1}$, the algorithm assigns the nodes in $\mathcal{D}_{2|c_1}$ and follows the same process until all nodes are assigned ($\mathcal{D}_{1|c_1}$ to $\mathcal{D}_{k|c_1}$).

[1] We refer to nodes in $\mathcal{G}_1$ interchangeably using their vertex id or binary index

Algorithm 1: Gray Optimizer.

Input : G_1; G_2; c_1; v_r

Assign v_r to c_1
for $i\ in [1:k]$ **do**
  Initialize $\mathcal{H}_1, \mathcal{H}_2 = \emptyset$
  $\mathcal{H}_1 \leftarrow \{\binom{k}{i}$ non-assigned nodes in G_2 with the highest probability$\}$
  for $c_j \in \mathcal{D}_{i|c_1}$ **do**
    Calculate $p(l_j/c_j) = \prod_{v \in l_j/c_j} p(v)$
    $\mathcal{H}_2 \leftarrow p(l_j/c_j)$
  end
  Match vertices in $\mathcal{H}_1$ to $\mathcal{H}_2$ based on Hungarian algorithm.
end

The assignment objective in stage i of the process is to maximize $p(\mathcal{L}_i|v_r)$. Note that (3) can be written as:

$$\sum_{i=1}^{k} Maximize\{p(\mathcal{L}_i|v_r)\}. \tag{4}$$

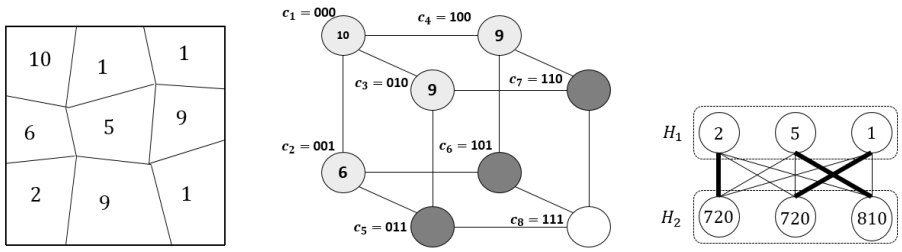
where $p(\mathcal{L}_i|v_r)$ represents the probability of all complete i-bit BRG cycles that include c_1 ($v_r \rightarrow c_1$). Consider one such a cycle as l. Based on the following lemma, there exists one and only one node c_j in l that has a Hamming distance of i from c_1, which means that $c_j \in \mathcal{D}_{i|c_1}$. Therefore, every complete i-bit BRG cycle given index c_1 includes one node in $\mathcal{D}_{i|c_1}$. On the other hand, every node in $\mathcal{D}_{i|c_1}$ corresponds to a unique complete i-bit BRG cycle passing through c_1, as results from Lemma 1. Therefore, all complete i-bit BRG cycles are considered in stage i and we maximize their probabilities in this stage of the assignment.

Lemma 2. *For each node c_i in a complete x-bit BRG cycle, there exists one and only one node with the Hamming distance of x from c_i.*

Proof. A complete x-bit BRG cycle includes 2^x nodes and only x bits are affected. Therefore, the only index that can exist with the Hamming distance of x from c_i is the one in which all x Hamming bits are opposite.

The assignment process in the stage i of GO creates a bipartite graph, i.e., $(\mathcal{H}_1, \mathcal{H}_2, \mathcal{E}_3)$, where $\mathcal{H}_1$ and $\mathcal{H}_2$ are two set of nodes, and $\mathcal{E}_3$ represents the set of edges. In this stage, the nodes in sets $\mathcal{D}_{1|c_1}$, $\mathcal{D}_{2|c_1}$,...,$\mathcal{D}_{i-1|c_1}$ are already assigned and we aim to find the best assignment for the nodes in $\mathcal{D}_{i|c_1}$ such that $p(\mathcal{L}_i|v_r)$ is maximized. Among the remaining nodes in $\mathcal{V}$, we choose $\binom{k}{i}$ of them that have the highest probabilities, as $|\mathcal{D}_{i|c_1}| = \binom{k}{i}$, and allocate them to $\mathcal{H}_1$.

On the other hand, for each node c_j in $\mathcal{D}_{i|c_1}$, we construct the unique complete i-bit BRG cycle including c_j and c_1. Let us represent this cycle by l_j. Note that all nodes included in l_j are assigned except c_j. The algorithm calculates the



(a) Sample grid. (b) Graph $G_1(\mathcal{C}, \mathcal{E}_1)$. (c) Graph $G_2(\mathcal{V}, \mathcal{E}_2)$.

Fig. 3: An example of embedding graphs generated based on a sample grid.

probability of the set of nodes in l_j excluding c_j and allocates it to a node in $\mathcal{H}_2$. Based on (1), this probability can be calculated as:

$$p(l_j \setminus \{c_j\}) = \prod_{v \in l_j \setminus \{c_j\}} p(v), \tag{5}$$

The algorithm repeats the process for all nodes in $\mathcal{D}_{i|c_1}$.Next, the best matching is found between these two sets of nodes based on the Hungarian algorithm [12].

Lemma 3. *In stage i, GO maximizes $p(\mathcal{L}_i|v_r)$ given the currently assigned nodes ($\mathcal{D}_{1|c_1}$, $\mathcal{D}_{2|c_1}$, ..., $\mathcal{D}_{i-1|c_1}$).*

Proof. We prove the lemma based on mathematical induction.
Base case: For $i = 1$, given that the node v_r is assigned to c_1, we aim to prove that GO maximizes $p(\mathcal{L}_1|v_r)$. To start with, GO chooses $\binom{k}{1}$ remaining nodes of $\mathcal{V}$ for the purpose of assignment. The optimal assignment of nodes in $\mathcal{D}_{1|c_1}$ is a permutation of the chosen nodes; otherwise, they could be replaced with a node with a higher probability that would result in a higher value for $p(\mathcal{L}_1|v_r)$. Next, the algorithm generates a bipartite graph $(\mathcal{H}_1, \mathcal{H}_2, \mathcal{E}_3)$. The probability of chosen nodes are allocated to $\mathcal{H}_1$, and the nodes in $\mathcal{H}_2$ represent the probability of complete 1-bit gray cycles constructed from $c_j \in \mathcal{D}_{1|c_1}$ and the node c_1, excluding the probability of c_j itself. As the Hungarian algorithm finds the best match between nodes, GO results in maximal $p(\mathcal{L}_1|v_r)$ given the node c_1.

Induction step: Let us assume that GO has maximized the probabilities of complete x-bit BRG cycles for $x = 1$ to $i - 1$ in stages one to $i - 1$. We prove that in stage i, the algorithm maximizes complete i-bit gray cycles, given the previously assigned nodes.

Based on Lemma 2, all complete i-bit BRG cycles are considered in stage i, as each such cycle includes exactly one node in $\mathcal{D}_{i|c_1}$, which has the highest Hamming distance from c_1. *GO* starts by choosing the cells with the highest probabilities and assigning them to $\mathcal{H}_1$. Same as in the base case, we know that the optimal assignment in this stage includes the chosen set of nodes. Next, the nodes in $\mathcal{H}_2$ are assigned based on finding the probability of complete i-bit BRG cycles for nodes in $\mathcal{D}_{i|c_1}$, excluding the nodes themselves from the probability. As the Hungarian algorithm results in an optimal match, the best permutation of nodes in $\mathcal{H}_1$ is matched to complete i-bit BRG cycles. □

4 Scaling Up Gray Optimizer

The time complexity of the Hungarian algorithm is shown to be $\mathcal{O}(n^3)$ [12], which results in the complexity of $\mathcal{O}(n(\log_2 n)^4)$ for GO. The algorithm can lead to significant improvements in the processing of HVE operations; however, there are two major drawbacks once the algorithm is applied to grids with high granularities. *(i)* The complexity of the algorithm creates a processing time bottleneck for its application in HVE; *(ii)* The calculation of probabilities for large complete BRG cycles may result in numerical inaccuracies. To make *GO* applicable to grids with higher levels of granularity, we propose two variations.

The first proposed algorithm, called *Multiple Seed Gray Optimizer (MSGO)* (Section 4.1), generates non-overlapping clusters and applies *GO* within each one of them. The second algorithm, called Scaled Gray Optimizer (SGO) (Section 4.2) takes a *Breadth-First Search (BFS)* [15] approach. The performance of BFS is preferred to its counterpart *Depth-First Search (DFS)* as the nodes closer to the seed have higher probabilities. Thus, it is reasonable to consider those nodes earlier in the process.

4.1 Multiple Seed Gray Optimizer (MSGO)

The starting point of the *GO* algorithm, which we refer to as *seed*, was chosen as the node in G_2 with the maximum probability. However, the algorithm can work starting with any initial seed, then follow the assignment process for other nodes in ascending order of their Hamming distance from the seed. Furthermore, as BRG cycles become larger, their associated probability becomes smaller. Thus, one way to reduce the complexity of *GO* is to run the algorithm up to a particular *depth*. Essentially, the algorithm aims at optimizing BRG cycles up to a certain length. We enhance *GO* by running Algorithm 1 with multiple seeds, and also by limiting the depth of the assignment.

Definition 4 *Depth: For a given seed c_j, the GO algorithm is said to run with a depth of i if it only considers the assignment of nodes in $\mathcal{D}_{1|c_j}$, $\mathcal{D}_{2|c_j}$, ..., $\mathcal{D}_{i|c_j}$.*

The pseudocode of the proposed approach is presented in Algorithm 2. The algorithm starts by assigning the node with the highest probability in G_2 to the origin of G_1 or a random index. However, instead of running GO with respect to this index for all depths from one to k, MSGO runs GO with the specified depth as input. The algorithm completes the process of assignment for a cluster of indices in G_1. MSGO then chooses a random index of G_1 among the remaining indices and assigns it to the node in G_2 with maximum probability among remaining nodes. Similarly, this index is used as a seed for GO with the specified depth and generates a new cluster. The cluster-based approach continues until all nodes are assigned to an index. The algorithm supports variable cluster sizes based on the underlying application.

The MSGO algorithm provides a robust solution for grids with higher granularity. The algorithm no longer suffers the drawbacks of GO when the grid size grows, such as numerical inaccuracies in the calculation of the probability of large cycles. The complexity of the algorithm depends on the depth chosen as input, and in low depths, it can be implemented in $\mathcal{O}(n(\log_2 n))$. MSGO can

Algorithm 2: Multiple Seed Gray Optimizer (MSGO).

Input : G_1; G_2; *depth*

1 Select a random index on G_1 which is not currently assigned
2 Assign the index with the node that has the maximum probability in G_2
3 Apply Algorithm 1 on the selected index with the specified depth
4 **Repeat** lines 1-3 **until** all indices are assigned

significantly reduce the number of operations required for the implementation of HVE in location-based alert systems, and therefore, making it a practical solution for preserving the privacy of users in location-based alert systems.

4.2 Scaled Gray Optimizer (SGO)

SGO considers overlapping clusters and necessitates that all nodes act as seed during the assignment process. The pseudocode of the proposed approach is presented in Algorithm 3. SGO starts by assigning the node with the highest probability to an index on G_1. However, instead of assigning indices with all depths from one to k with respect to index c_1, the SGO algorithm runs GO with the depth of one. Next, SGO sorts the indices in $\mathcal{D}_{1|c_1}$ based on their assigned probabilities in descending order and runs GO on each one of them. Once the algorithm is applied on all the indices in $\mathcal{D}_{1|c_1}$, the process repeats for indices in $\mathcal{D}_{2|c_1}$, $\mathcal{D}_{3|c_1}$, ..., etc. The algorithm continues until all indices are assigned to a node.

Algorithm 3: Scaled Gray Optimizer (SGO).

Input : G_1; G_2

1 Assign the cell with the highest probability to the origin of G_1, i.e., c_1
2 Apply Algorithm 1 on c_1 with the depth of one
3 **for** i *in* $[1:k]$ **do**
4 | Sort $\mathcal{D}_{i|c_1}$ in descending order of probabilities assigned to its indices
5 | **for** c_j *in* $\mathcal{D}_{i|c_1}$ **do**
6 | | Apply Algorithm 1 on c_j with the depth of one
7 | **end**
8 **end**

Example 3 *Consider a map of a vast rural area in which there is a likelihood of bush fire or security concerns for the residents. Therefore, the alert based notification system is implemented by a service provider to notify the farmers on their request. Due to a large number of cells included in the map, GO and MSGO require a comparably high computation overhead. Hence, the SGO algorithm could be used to improve the time complexity. Starting with the cell that has the highest probability, SGO executes GO with the minimum depth, i.e., one. Then, in a*

breadth-first-search manner, the algorithm moves to the first neighbors of that node, taking them as seeds of GO, prioritizing the nodes with higher assigned probability. The process continues until all the nodes of the grid are assigned.

The complexity of the SGO algorithm is $\mathcal{O}(n(\log_2 n))$, which enables GO to be applied on grids with higher granularity. SGO can significantly reduce the number of operations required for the implementation of HVE in location-based alert systems, and therefore, making it a practical solution for preserving the privacy of users in location-based alert systems.

5 Experimental Evaluation

5.1 Experimental Setup

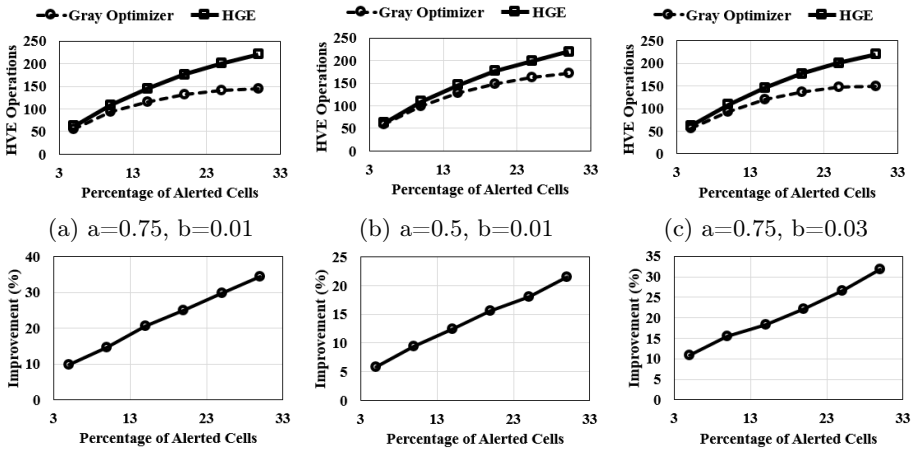
We conduct our experiments on a 3.40GHz core-i7 Intel processor with 8GB RAM running 64-bit Windows 7 OS. The code is implemented in Python, and we used the LogicMin Library [7] for binary minimization of token expressions. We compare the proposed approaches (GO, MGSO and SGO) against the hierarchical Gray encoding technique from [10] (labeled *HGE*), the state-of-the-art in location alerts on HVE-encrypted data.

To model the probability of partition cells becoming alert zones, we use the sigmoid function $\mathcal{S}(x) = 1/(1 + \exp^{-b(x-a)})$, where a and b are parameters controlling the function shape. The output value is between zero and one. The sigmoid function is a frequent model used in machine learning, and we choose it because we expect that, in practice, the probability of individual cells becoming part of an alert zone can be computed using such a model built on a regions' map of features (e.g., type of terrain, building designation, point-of-sale information, etc). Parameter a of the sigmoid controls the *inflection* point of the curve, whereas b controls the gradient. Fig. 5 plots the logistic function for several different values of a and b which we use in our evaluation.

5.2 Gray Optimizer Evaluation

GO is our core proposed algorithm to reduce the number of HVE operations required to support alert zones. Specifically, by *HVE operations* we refer to the computation executed by the server to determine matches between tokens and encrypted user locations. Recall that, for each non-star item in a token, a number of expensive bilinear map operations are required. GO aims to minimize the number such non-star items in tokens by choosing an appropriate encoding of the domain. Our comparison benchmark is the approach from [10] which uses a hierarchical quadtree structure to partition the data domain. In our experiments, we refer to this approach as *HGE*, and we present our result as an improvement in terms of computation overhead compared with [10].

Improvement in HVE Operations Fig 4 summarizes the evaluation results of GO for three logistic function parameter settings. The grid size is set to 100 cells (recall from our earlier discussion that GO can only support relatively low granularities). Fig. 4 shows the total number of bilinear pairings performed for



(a) a=0.75, b=0.01 (b) a=0.5, b=0.01 (c) a=0.75, b=0.03

(d) a=0.75, b=0.01 (e) a=0.5, b=0.01 (f) a=0.75, b=0.03

Fig. 4: Evaluation of GO, grid size = 100 cells.

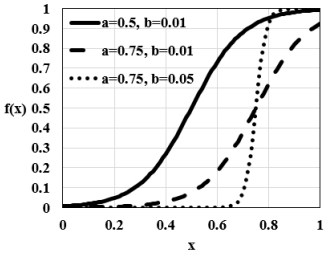


Fig. 5: Sigmoid function shown for parameters used in the experiments.

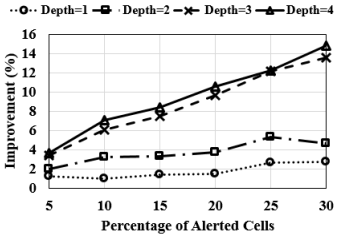
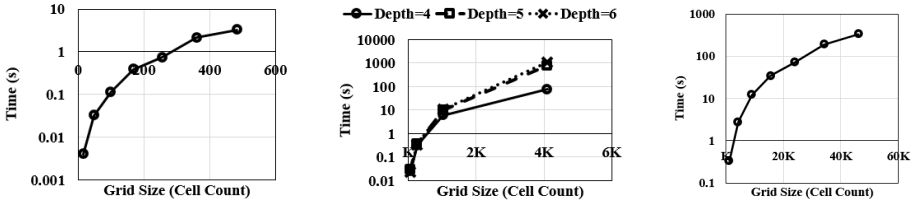


Fig. 6: Performance evaluation of GO for varying depth (100 cells).

a ciphertext-token pair. GO clearly outperforms the approach from [10]. The relative gain in performance of GO increases when the size of the alert zone increases (i.e., when there are more grid cells covered by the alert zone). This can be explained by the fact that a larger input set gives GO more flexibility to optimize the encoding and decrease the number of non-star entries in a token. In terms of percentage gains, GO can improve performance by up to 40%, which is quite significant. Also, note that the gains are significant for all parameters of the sigmoid function used. In general, we identified that a higher a value leads to more pronounced gains. This is an encouraging factor, because a higher a corresponds to a more skewed probability case, where a relatively small number of cells are more likely to be included in an alert zone than others. In practice, one would expect that to be the case, since events that trigger alerts also tend to be concentrated over a relatively small area (e.g., very popular hotspots, certain facilities that present higher risks, like a chemical plant, etc.).

Impact of Depth Recall that the reduction in computation achieved by GO depends on the depth at which the algorithm is run (GO works similar to a



(a) Gray Optimizer. (b) MSGO. (c) SGO.

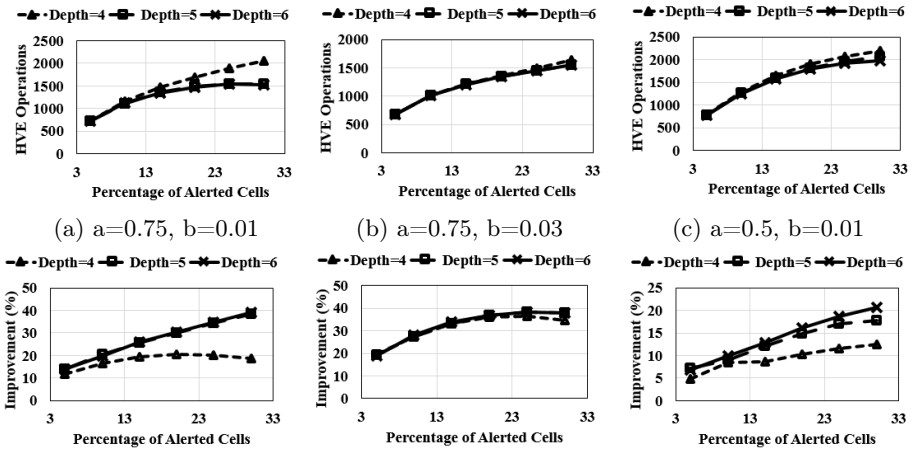
Fig. 7: The execution time performance.

depth-first search graph algorithm). In general, running the algorithm with a higher depth will produce better results in terms of performance gain at runtime (i.e., when matching is performed at the server), but it also requires a lot more computational time to compute a good encoding (which is a one-time cost). Fig. 6 captures the impact of depth on improvement. In this experiment, GO is executed on a single cell with different depths, and the remaining cells are assigned randomly (the experiment is specifically designed to show the effect of using lower depths on GO). As expected, there is a clear increasing trend, with higher depths resulting in better improvement factors. However, after a sharp initial gain (illustrated by the large distance between the chart graphs corresponding to depths 2 and 3), the improvement stabilizes, and it may no longer be worth increasing the depth of the computation considerably (the gains are stabilizing between depths 3 and 4).

Execution Time Fig. 7a illustrates the execution time of GO. Recall that, the execution time of GO is influenced by the granularity of the grid (finer granularities increase execution time). The results show that GO can complete within a short execution time for smaller grid sizes; however, as the grid granularity increases, there is a sharp increase in execution time. Therefore, GO may not be practical to apply for high granularity grids, and that is the main motivation behind our two variations, MSGO and SGO (which are evaluated next). Moreover, as the grid granularity increases, the length of cycles becomes larger, which will also result in numerical inaccuracies when executing GO. The execution time required by GO for values up to 600 cells is around 10 seconds. We observed that this value is the maximum number of cells for which GO performs reasonably; beyond this level, the algorithm is not suitable due to increased execution time and numerical inaccuracies associated with the calculation of probabilities for large cycles.

5.3 Evaluation of GO Variations on Higher Granularity Grids

As discussed previously, GO does not perform well when directly applied to high granularity grids. To improve the computational complexity of GO, we proposed two extensions of the algorithm, namely, MSGO and SGO. Next, we evaluate experimentally both these variations.



(a) a=0.75, b=0.01 (b) a=0.75, b=0.03 (c) a=0.5, b=0.01

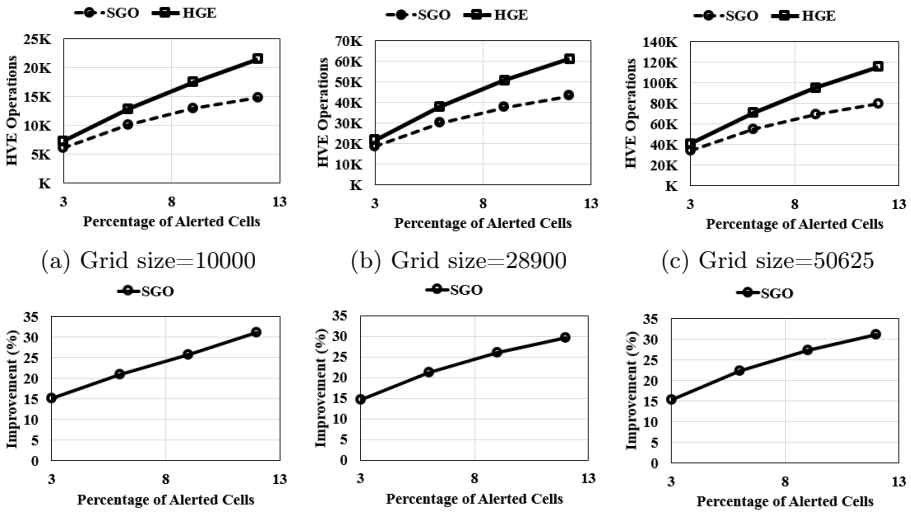
(d) a=0.75, b=0.01 (e) a=0.75, b=0.03 (f) a=0.5, b=0.01

Fig. 8: Performance evaluation of MSGO (grid size = 1024 cells).

MSGO Fig 8 illustrates the performance of MSGO compared to the HGE benchmark scheme from [10]. Unlike the single seed GO, we are able to evaluate the performance of MSGO for grids with much higher granularity (i.e., 1024 cells in this case). There is a similar trend in terms of gain as we have observed with GO, where larger alert zones provide more opportunities for advantageous encodings, and thus overall performance is improved (the percentage of HVE operations eliminated is higher). The relative gain obtained is very close to 50% compared to the benchmark. Also, the absolute amount of improvement is better than for GO in all cases. This occurs due to the fact that MSGO can support higher-granularity grids, and in this setting there is more flexibility in choosing a good encoding (due to the larger number of cells, there are significantly more choices for our algorithm). As expected, increasing the depth of MSGO leads to higher improvement percentage, but the trade-off is a larger computation complexity.

Comparing Figs. 4 and 8, we remark that the MSGO algorithm obtains similar performance gains as the core algorithm GO for low granularity grids, but with a much lower computational overhead. For high granularity grids, GO cannot keep up in terms of computational overhead, whereas MSGO scales reasonably well, and it is able to still obtain significant improvements. One main reason is that MSGO no longer requires the calculation of probabilities of large cycles, avoiding numerical inaccuracies and reducing overall computational overhead. The complexity of the algorithm can be as low as $\mathcal{O}(n(\log_2 n))$ depending on the chosen depth value, which provides a robust and efficient solution for reducing the number of HVE operations.

The execution time of MSGO is illustrated in Fig. 7b. The graph indicates that even for a high level of granularity, such as $4,000$, the algorithm requires less than 15 minutes to encode the grid, depending on the specified depth at the input. As expected, by increasing the depth of the algorithm, better performance



(a) Grid size=10000 (b) Grid size=28900 (c) Grid size=50625

(d) Grid size=10000 (e) Grid size=28900 (f) Grid size=50625

Fig. 9: SGO performance evaluation, varying grid size.

can be achieved in terms of HVE operations, at the cost of higher computational overhead. The MSGO algorithm can be extended for an arbitrary number of cells on the grid, and also it may have various cluster sizes depending on the application.

SGO Fig 9 illustrates the performance gain obtained by SGO. In this experiment, we focused on applying the algorithm to much larger number of cells, up to $50,625$ (which is equivalent to a 225×225 square grid). Similar to the MSGO algorithm, the improvement achieved by SGO occurs even when the alert zones are small. Since the overall number of cells is larger, the SGO algorithm has even more flexibility in choosing an advantageous encoding, resulting in strong performance gains. For example, at 9% ratio of alert cells, the SGO algorithm results in 25.8, 26, and 27.3% improvements for grid sizes of $10,000$, $28,900$, and $50,625$, respectively.

The execution time of SGO is shown in Fig. 7c. Even for very large grid sizes, such as $50,625$, the algorithm requires less than six minutes to encode the grid. Therefore, the system can be set to regularly update the probabilities and run the algorithm at six-minute intervals, if needed. To compare this time performance with GO, consider the maximum grid size for which the encoding can be computed within 60 seconds in each case. As shown in Fig. 7a, this number corresponds to a grid size of 1200 for GO, whereas in a similar time, SGO can be applied on the grid size of $22,000$ cells. Therefore, the SGO algorithm requires significantly lower computation overhead to execute compared with GO and even MSGO algorithms, while the performance gain in terms of HVE operations reductions is still solid.

6 Related Work

Location Privacy. Preserving the privacy of users in communication networks and online platforms has been one of the most challenging research problems in the past two decades. In the widely accepted scenario, users provide their location to service providers in exchange for location-based services they offer. The goal is to provide the service without user privacy being compromised by any of the parties involved. Early works to tackle this problem were focused on hiding or obfuscating user locations to achieve a privacy metric termed as k-anonymity. The location of a user is said to be k-anonymous if it is not distinguishable from at least $k-1$ other queried locations [22].

In [13], the authors aim to provide k-anonymity by hiding the location of user among $k-1$ fake locations and requesting for desired services for all k locations at the same time. The generation of such dummy locations based on a virtual grid or circle was considered in [18]. The authors in [17] conducted the selection of dummy locations predicated on the number of queries made on the map and aimed at increasing the entropy of k locations in each set. In [5], random regions that enclose the user locations were introduced to bring uncertainty in the authentication of user locations. Unfortunately, fake locations can be revealed particularly in trajectories and with the existence of prior knowledge about the map and users.

Later on, approaches based on *Cloaking Regions (CRs)* proposed by [11] gained momentum in the literature. The principal idea behind this method is to use a trusted anonymizer that clusters k real user locations and query the area they are enclosed by to retrieve points of interest. Doing so, CRs aim to achieve k-anonymity for users and preserve their privacy. This approach is partially effective when snapshots of trajectories are considered, but once users are seen in trajectories, their location privacy would be severely at risk [19]. Even for individual snapshots, it must be noted that a coarse area of real locations is released to the service provider, which could threaten the location privacy of users. Moreover, the CR-based approaches are susceptible to inference attacks predicated on the background knowledge or so-called side information. One such side information is the knowledge about the number of queries made on different locations of the map [17].

More recently, a model for privacy preservation in statistical databases termed as *differential privacy* was developed in [8]. The metric provides a promising prospect for aggregate queries; however, it is not suitable for private retrieval of specific data from datasets. Closer to HVE approach, a private information protocol was proposed in [9]. The PIR technique is based on cryptography and shown to be secure for private retrieval of information. Despite the promising results, there exists an assumption behind PIR approach that the user already knows about the points of interest. Therefore, PIR is not suitable for location-based alert systems as users are not aware of alert zone whereabouts.

Searchable Encryption. Originated from works such as [21], the concept of search encryption was proposed to provide a secure cryptographic search of keywords. Initially, only the exact matches of keywords were supported and later on

the approach was extended for comparison queries in [2], and to subset queries and conjunctions of equality in [3]. The authors in [3] also proposed the concept of HVE, used as the underlying tool to provide a secure location-based alert system. This approach and its extension in [1] preserves the privacy of encrypted messages and tokens with the overhead of high computational complexity. The authors in [10] introduced and adopted the HVE for location-based alert systems, conducting the predicate match at a trusted provider, preserving the privacy of encrypted messages as well as tokens. Despite the promising results of the approach for privacy preservation in location-based alert systems, further reduction of computational overhead is necessary to increase the practicality.

7 Conclusion

We proposed a technique to reduce the computational overhead of HVE predicate evaluation in location-based alert systems. Specifically, we used graph embeddings to find advantageous domain space encodings that help reduce the required number of expensive HVE operations. Our heuristic solutions offer an improvement in computation overhead of 50% compared to existing work, and they can scale to domain partitionings of fine granularity. In future work, we will focus on deriving cost models that can estimate the amount of savings in HVE overhead based on the distribution and frequency of alert events. We will also investigate extending the graph embedding approach to other types of searchable encryption, beyond HVE (e.g., Inner Product Evaluation).

Acknowledgment. This research has been funded in part by NSF grants IIS-1910950 and IIS-1909806, the USC Integrated Media Systems Center (IMSC), and unrestricted cash gifts from Microsoft and Google . Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the sponsors such as the National Science Foundation. Note: Cyrus Shahabi receives consulting income from Google for a different unrelated project.

References

1. Blundo, C., Iovino, V., Persiano, G.: Private-key hidden vector encryption with key confidentiality. In: International Conference on Cryptology and Network Security. pp. 259–277. Springer (2009)
2. Boneh, D., Sahai, A., Waters, B.: Fully collusion resistant traitor tracing with short ciphertexts and private keys. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 573–592. Springer (2006)
3. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Theory of Cryptography Conference. pp. 535–554. Springer (2007)
4. Chandrasekharam, R., Vinod, V., Subramanian, S.: Genetic algorithm for embedding a complete graph in a hypercube with a vlsi application. Microprocessing and microprogramming **40**(8), 537–552 (1994)
5. Cheng, R., Zhang, Y., Bertino, E., Prabhakar, S.: Preserving user location privacy in mobile data management infrastructures. In: International Workshop on Privacy Enhancing Technologies. pp. 393–412. Springer (2006)

6. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security **19**(5), 895–934 (2011)
7. Drake, C.: Two-level logic minimization (2012), https://pyeda.readthedocs.io/en/latest/2llm.html
8. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Theory of cryptography conference. pp. 265–284. Springer (2006)
9. Ghinita, G., Kalnis, P., Khoshgozaran, A., Shahabi, C., Tan, K.L.: Private queries in location based services: anonymizers are not necessary. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 121–132 (2008)
10. Ghinita, G., Rughinis, R.: An efficient privacy-preserving system for monitoring mobile users: making searchable encryption practical. In: Proceedings of the 4th ACM conference on Data and application security and privacy. pp. 321–332. ACM (2014)
11. Gruteser, M., Grunwald, D.: Anonymous usage of location-based services through spatial and temporal cloaking. In: Proceedings of the 1st international conference on Mobile systems, applications and services. pp. 31–42 (2003)
12. Jonker, R., Volgenant, A.: A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing **38**(4), 325–340 (1987)
13. Kido, H., Yanagisawa, Y., Satoh, T.: An anonymous communication technique using dummies for location-based services. In: ICPS'05. Proceedings. International Conference on Pervasive Services, 2005. pp. 88–97. IEEE (2005)
14. Lai, S., Patranabis, S., Sakzad, A., Liu, J.K., Mukhopadhyay, D., Steinfeld, R., Sun, S.F., Liu, D., Zuo, C.: Result pattern hiding searchable encryption for conjunctive queries. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 745–762 (2018)
15. Leiserson, C.E., Rivest, R.L., Cormen, T.H., Stein, C.: Introduction to algorithms, vol. 6. MIT press Cambridge, MA (2001)
16. Nguyen, K., Ghinita, G., Naveed, M., Shahabi, C.: A privacy-preserving, accountable and spam-resilient geo-marketplace. In: Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. pp. 299–308. ACM (2019)
17. Niu, B., Li, Q., Zhu, X., Cao, G., Li, H.: Achieving k-anonymity in privacy-aware location-based services. In: IEEE INFOCOM 2014-IEEE Conference on Computer Communications. pp. 754–762. IEEE (2014)
18. Niu, B., Zhang, Z., Li, X., Li, H.: Privacy-area aware dummy generation algorithms for location-based services. In: 2014 IEEE International Conference on Communications (ICC). pp. 957–962. IEEE (2014)
19. Shaham, S., Ding, M., Liu, B., Dang, S., Lin, Z., Li, J.: Privacy preserving location data publishing: A machine learning approach. IEEE Transactions on Knowledge and Data Engineering (2020)
20. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: 2000 IEEE Symposium on Security and Privacy. pp. 44–55. IEEE (2000)
21. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000. pp. 44–55. IEEE (2000)
22. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **10**(05), 557–570 (2002)

A Primer on HVE Encryption

Hidden Vector Encryption (HVE) [3] is a searchable encryption system that supports predicates in the form of conjunctive equality, range and subset queries. Search on ciphertexts can be performed with respect to a number of *index attributes*. HVE represents an attribute as a bit vector (each element has value 0 or 1), and the search predicate as a *pattern* vector where each element can be 0, 1 or '*' that signifies a wildcard (or "don't care") value. Let l denote the HVE *width*, which is the bit length of the attribute, and consequently that of the search predicate. A predicate evaluates to $True$ for a ciphertext C if the attribute vector I used to encrypt C has the same values as the pattern vector of the predicate in all positions that are not '*' in the latter. Fig. 2c illustrates the two cases of *Match* and *Non-Match* for HVE.

HVE is built on top of a symmetrical bilinear map of composite order [3], which is a function $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ such that $\forall a, b \in G$ and $\forall u, v \in \mathbb{Z}$ it holds that $e(a^u, b^v) = e(a, b)^{uv}$. $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic multiplicative groups of composite order $N = P \cdot Q$ where P and Q are large primes of equal bit length. We denote by $\mathbb{G}_p$, $\mathbb{G}_q$ the subgroups of $\mathbb{G}$ of orders P and Q, respectively. Let l denote the HVE *width*, which is the bit length of the attribute, and consequently that of the search predicate. HVE consists of the following phases:

Setup. The TA generates the public/secret (PK/SK) key pair and shares PK with the users. SK has the form:

$$SK = (g_q \in \mathbb{G}_q, \quad a \in \mathbb{Z}_p, \quad \forall i \in [1..l] : u_i, h_i, w_i, g, v \in \mathbb{G}_p)$$

To generate PK, the TA first chooses at random elements $R_{u,i}, R_{h,i}, R_{w,i} \in \mathbb{G}_q, \forall i \in [1..l]$ and $R_v \in \mathbb{G}_q$. Next, PK is determined as:

$$PK = (g_q, \quad V = vR_v, \quad A = e(g, v)^a,$$

$$\forall i \in [1..l] : U_i = u_i R_{u,i}, \quad H_i = h_i R_{h,i}, \quad W_i = w_i R_{w,i})$$

Encryption uses PK and takes as parameters index attribute I and message $M \in \mathbb{G}_T$. The following random elements are generated: $Z, Z_{i,1}, Z_{i,2} \in \mathbb{G}_q$ and $s \in \mathbb{Z}_n$. Then, the ciphertext is:

$$C = (C^{'} = MA^s, \quad C_0 = V^s Z,$$

$$\forall i \in [1..l] : C_{i,1} = (U_i^{I_i} H_i)^s Z_{i,1}, \quad C_{i,2} = W_i^s Z_{i,2})$$

Token Generation. Using SK, and given a search predicate encoded as pattern vector I_*, the TA generates a search token TK as follows: let J be the set of all indices i where $I_*[i] \neq *$. TA randomly generates $r_{i,1}$ and $r_{i,2} \in \mathbb{Z}_p, \forall i \in J$. Then

$$TK = (I_*, K_0 = g^a \prod_{i \in J} (u_i^{I_*[i]} h_i)^{r_{i,1}} w_i^{r_{i,2}},$$

$$\forall i \in [1..l] : K_{i,1} = v^{r_i,1}, \quad K_{i,2} = v^{r_i,2})$$

Query is executed at the server, and evaluates if the predicate represented by TK holds for ciphertext C. The server attempts to determine the value of M as

$$M = C^{'}/(e(C_0, K_0)/\prod_{i \in J} e(C_{i,1}, K_{i,1})e(C_{i,2}, K_{i,2}) \tag{6}$$

If the index I based on which C was computed satisfies TK, then the actual value of M is returned, otherwise a special number which is not in the valid message domain (denoted by $\perp$) is obtained.