A secure location-based alert system with tunable privacy-performance trade-off

Gabriel Ghinita[1] (ID) · Kien Nguyen[2] · Mihai Maruseac[1] · Cyrus Shahabi[2]

Abstract

Monitoring location updates from mobile users has important applications in many areas, ranging from public health (e.g., COVID-19 contact tracing) and national security to social networks and advertising. However, sensitive information can be derived from movement patterns, thus protecting the privacy of mobile users is a major concern. Users may only be willing to disclose their locations when some condition is met, for instance in proximity of a disaster area or an event of interest. Currently, such functionality can be achieved using *searchable encryption*. Such cryptographic primitives provide provable guarantees for privacy, and allow decryption only when the location satisfies some predicate. Nevertheless, they rely on expensive *pairing-based cryptography (PBC)*, of which direct application to the domain of location updates leads to impractical solutions. We propose secure and efficient techniques for private processing of location updates that complement the use of PBC and lead to significant gains in performance by reducing the amount of required pairing operations. We implement two optimizations that further improve performance: materialization of results to expensive mathematical operations, and parallelization. We also propose an heuristic that brings down the computational overhead through enlarging an alert zone by a small factor (given as system parameter), therefore trading off a small and controlled amount of privacy for significant performance gains. Extensive experimental results show that the proposed techniques significantly improve performance compared to the baseline, and reduce the searchable encryption overhead to a level that is practical in a computing environment with reasonable resources, such as the cloud.

✉ Gabriel Ghinita
    gghinita@cs.umb.edu

    Kien Nguyen
    kien.nguyen@usc.edu

    Mihai Maruseac
    mmarusea@cs.umb.edu

    Cyrus Shahabi
    shahabi@usc.edu

[1] University of Massachusetts, Boston, MA USA

[2] University of Southern California, Los Angeles, CA USA

1 Introduction

Modern mobile devices with positioning capabilities (e.g., GPS) allow users to be informed about events that occur in their proximity. Many classes of applications benefit from the large-scale availability of location data, ranging from public health (e.g., COVID-19 contact tracing) and national security to social networks and advertising. One particular scenario of interest is that of *location-based alert systems*, where mobile users wish to be immediately notified when their current location satisfies some conditions, expressed as a spatial *search predicate*. For instance, in a public safety scenario, users want to be notified when they are getting close to a dangerous accident area. Alternatively, in the commercial domain, a user may want to be alerted when a nearby sale event is underway.

The typical architecture of such a system uses a server that collects location updates from the users and checks whether the alert condition is met. Such a service is often provided by a commercial entity that is not fully trusted. The collection of user trajectories at a commercial site introduces serious privacy concerns, as sensitive personal information may be derived from a person's whereabouts [16, 20]. Therefore, protecting the privacy of users is a necessary feature of such a system, and the users must not report their exact locations to the server. Ideally, the only information that the server should be able to derive from the user updates is whether the conditions that the users subscribe to are satisfied or not. Syntactic privacy models [15, 17, 20, 34] that perform generalization of locations before sharing have been proven vulnerable, especially in the presence of background knowledge [16]. Furthermore, semantic privacy models such as differential privacy [9, 11, 12] are only suitable for releasing statistics, but not for processing privately individual updates.

Recently, several advanced encryption functions that allow evaluation of predicates on ciphertexts have been proposed [2, 6, 30]. These functions are broadly referred to as *searchable encryption (SE)* functions, since they allow the evaluation of certain types of queries without requiring decryption. Some of these encryption systems are asymmetric, i.e., they employ a secret key *SK* and a public key *PK* pair.

Figure 1 shows the envisioned system architecture, with three types of entities: *(i)* users who send encrypted location updates using the *PK* of a trusted authority; *(ii)* a *trusted authority (TA)* who generates *tokens* for spatial search predicates using secret key *SK*; and *(iii) the server (S)* that collects location updates from users and evaluates the predicates on
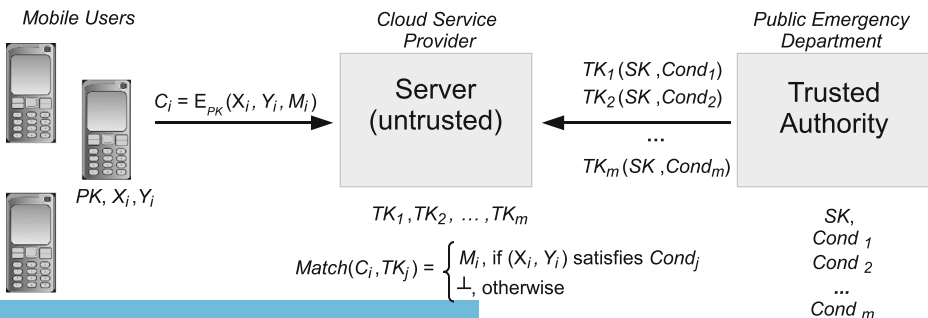
Fig. 1 Location-based Alert System

the ciphertexts using the tokens. In practice, the TA may represent the public emergency department of a city, which is responsible for the safety of the citizens. The TA is trusted, but it does not have the necessary infrastructure to support a large-scale alert system, hence it outsources this service to S.

However, S is a commercial entity that cannot be trusted with user locations, so the TA sets up a *SK/PK* pair, and distributes *PK* to the users. When an emergency occurs in a region, the TA creates a search token which is sent to S to be matched against the ciphertexts received from users. The properties of SE guarantee that S is able to evaluate the predicate on the ciphertext (e.g., whether the user location is enclosed in the region encoded by the search token) and learns only if the ciphertext matches or not, but no other information about user location.

To understand search on encrypted data, it helps to consider each ciphertext as being composed of two parts: an encrypted index I and encrypted message M. M is the payload of the ciphertext, just the same as in the case of conventional encryption. The novel part about searchable encryption is the presence of the index I, which is used for search, and can be seen as a parameter of the encryption function $E_{PK}(I, M)$. When a user u_i constructs its update, she uses her current coordinates (X_i, Y_i) as index, and performs encryption as $E_{PK}((X_i, Y_i), M_i)$. If the index satisfies the predicate specified by a token, then the server is able to recover the message M_i from the user. However, this does not imply that S can find the exact user location, as M_i may contain information that is of other nature (e.g., an emergency contact number).

One prominent approach to searchable encryption called *Hidden Vector Encryption (HVE)* was proposed in [6]. HVE can evaluate exact match, range and subset queries on ciphertexts. HVE uses bilinear maps on groups of composite order [19] as mathematical foundation and makes extensive use of expensive operations such as *bilinear map pairings*.

As a result, HVE is very expensive and scales poorly. Later in Section 6, we show that in order to process the update from a single user only, it may take up to 100 seconds. Clearly, direct application of HVE for alert systems is not suitable.

In this paper, we propose secure and efficient techniques to support private location-based alert systems using searchable encryption. To the best of our knowledge, this is the first study of applying asymmetric searchable encryption to the domain of private search with spatial predicates. Our specific contributions are:[1]

i   We devise specific constructions that allow application of HVE to the problem of location-based alert systems with a reduced number of *bilinear pairing* operations, thus lowering the computational overhead of HVE.

ii We develop optimizations based on reuse of expensive mathematical operation results and parallelization, which further reduce the HVE performance overhead.

iii We introduce a novel heuristic algorithm that provides effective means to tune the privacy-performance trade-off of the system, by allowing enlargement of alert zones by a small factor. By carefully enlarging the alert zone, one can obtain search tokens that require significantly smaller computation time to process.

iv We perform an extensive experimental evaluation which shows that the proposed approach brings the overhead of searchable encryption to acceptable levels in a computing environment such as the cloud.

---

[1]This submission is an extended version of [18]. Additional contributions consist of the technique for relaxation of alert zones presented in Section 5, and its evaluation in Section 6.4.

Section 2 overviews the proposed system and HVE. Section 3 presents the encoding techniques for efficient application of HVE, whereas Section 4 outlines the optimizations to reduce execution time.

In Section 5, we introduce the heuristic for privacy-performance trade-off tuning through alert zone enlargement.

Section 6 contains the experimental evaluation results, followed by a survey of related research in Section 7. Finally, Section 8 concludes the paper and highlights directions for future work.

2 Background

2.1 System and privacy model

Figure 2 illustrates the location-based alert system model, where n users $\{u_1, \ldots, u_n\}$ move within a two-dimensional domain. Users continuously report their coordinates and wish to be notified when their location falls within any of m *alert zones* $\{z_1, \ldots, z_m\}$. Alert zones (or simply *zones*) are defined by a trusted authority, as detailed later in this section. For simplicity, we assume that the space is partitioned by a regular grid of size $d \times d$, and each alert zone covers a number of grid cells. To facilitate presentation, we assume a square data domain, but our techniques can be immediately extended to a rectangular one, by adjusting the grid cell shape. The functional requirement of the system follows the spatial range query semantics, i.e., a user u must receive an alert corresponding to zone z if its location is *enclosed* by zone z.

The system (represented in Fig. 1) consists of three types of entities:

i **Mobile Users** subscribe to the alert system and periodically submit encrypted location updates.

ii 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.

iii The **Server (S)** is the provider of the alert system. It receives encrypted updates from users and search tokens from TA, and performs the predicate evaluation *Match* to decide whether encrypted location $C_i (1 \leq i \leq n)$ falls within alert zone j represented

$z_1 = \{F5, F6, F7, G5, G6, G7\}$

$z_2 = \{B6, B7\}$

$u_1 = \{B1\}$

$u_2 = \{G6\}$

$Match(C_1, TK_1) = \perp$

$Match(C_1, TK_2) = \perp$
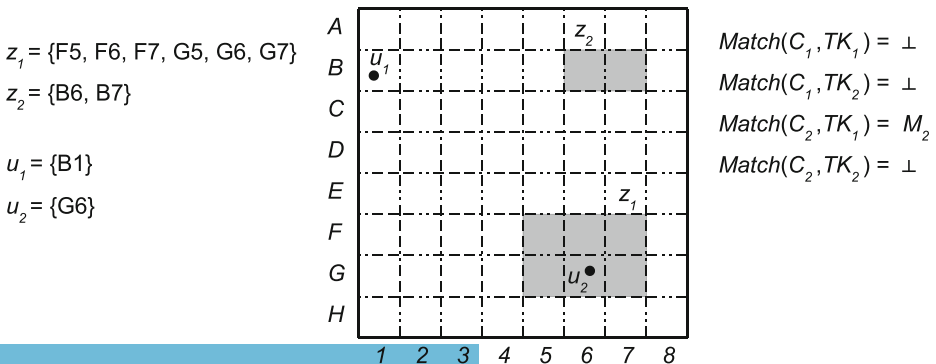
$Match(C_2, TK_1) = M_2$

$Match(C_2, TK_2) = \perp$

Fig. 2 System Model ($n = 2, m = 2, d = 8$)

by token $TK_j(1 \leq j \leq m)$. If the predicate holds, *Match* returns message M_i encrypted by the user, otherwise it returns a void message ($\perp$).

The *privacy requirement* 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, such as our motivating example in Section 1. 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 information about u's location.

In practice, the TA role is played by an organization such as a city's public emergency department. Such an actor is trusted not to disclose SK and compromise user privacy, but at the same time does not have the technological infrastructure to monitor a large user population. Hence, the alert service is outsourced to a commercial entity, e.g., a cloud provider that plays the role of the server. The TA will issue alert zones to signal that certain areas of the city are affected by an emergency.

A private location-based alert system is also useful in social networks. A social network user u can create a SK/PK pair and distribute PK to its buddies. Next, u creates a token that represents his/her current location, e.g., a downtown restaurant. The network provider (e.g., Facebook), plays the role of the server: it privately monitors users, and sends the identifiers of buddies in the downtown area back to u. No information is gained by the server about locations of non-matching users.

2.2 Searchable encryption with HVE

Hidden Vector Encryption (HVE) [6] is a searchable encryption system that supports predicates in the form of conjunctive equality, range and subset queries. Compared to earlier solutions [3, 5], HVE yields ciphertexts with considerably smaller length. 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
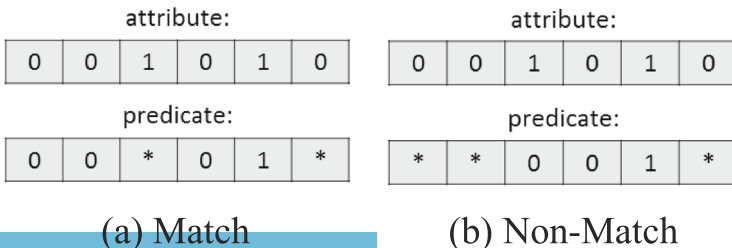

Fig. 3 Predicate evaluation on ciphertexts with HVE

predicate in all positions that are not '*' in the latter. Figure 3 illustrates the two cases of *Match* and *Non-Match* for HVE, whereas Algorithm 1 provides the matching pseudocode. We provide additional mathematical background on HVE encryption and its operations in Appendix A.

Algorithm 1 HVE_Match.

    Input   : HVE index $I = [I_1 : I_l]$
    Input   : HVE token $T = [T_1 : T_l]$
    Output: $True$ if I matches T, $False$ otherwise

1 **if** $\forall i \in [1 : l]$, $I_i = T_i$ or $T_i = *$ **then**
2 | return $True$
3 **else**
4 | return $False$

---

3 Proposed spatial HVE approaches

In Section 3.1 we outline a naive *baseline* technique which applies HVE in a straightforward manner to determine privately which users fall within one or more alert zones. The baseline leads to prohibitive costs, as shown by experiments in Section 6. To bring down the overhead of HVE, we propose in Section 3.2 a *hierarchical encoding* technique, which reduces the amount of cryptographic primitives (specifically, bilinear pairings) required during search. Next, in Section 3.3 we further refine hierarchical encoding and devise the *Gray encoding*, which achieves superior computation savings.

3.1 Baseline Encoding

Recall that the data space is partitioned by a two-dimensional $d \times d$ regular grid. When a user reports its position, it sends to the server an encryption of the grid cell it is enclosed by. Similarly, the TA defines the alert zones as a set of grid cells. Each grid cell can be uniquely identified by a *cell identifier*, with values between 1 and d^2. Thus, the straightforward way to support secure location-based alerts is to use an HVE index with width $l = d^2$. The data and query encoding are performed as follows:

– When user u enclosed by grid cell i reports its location, it uses a bitmap index I of width d^2 where all the bits are set to '0' except bit i which is set to '1'.
– The TA creates a single token for search, which captures all the alert zones. The token is a bitmap with d^2 bits where all bits corresponding to cell identifiers that are included in an alert zone are set to '*'. All other bits are set to '0'.
– At the server (i.e., at query time), according to the rules for HVE query evaluation from Section 2.2, a user will be determined as a $Match$ if and only if the '1' bit in the encrypted location will correspond to a '*' entry in the token.

Consider the example in Fig. 4, where $d = 3$. We have nine grid cells, so the width of the HVE is $l = 9$. There are two alert zones: z_1 which consists of a single grid cell (3), and z_2 which spans two grid cells (8 and 9). Two users report their locations: u_1 enclosed by cell 1, and u_2 enclosed by cell 8. The index vectors of the two users are shown in the diagram. A single token is used to represent both alert zones, and a '*' is placed in the positions corresponding to the cells enclosed by the zones, namely 3, 8 and 9. The predicate
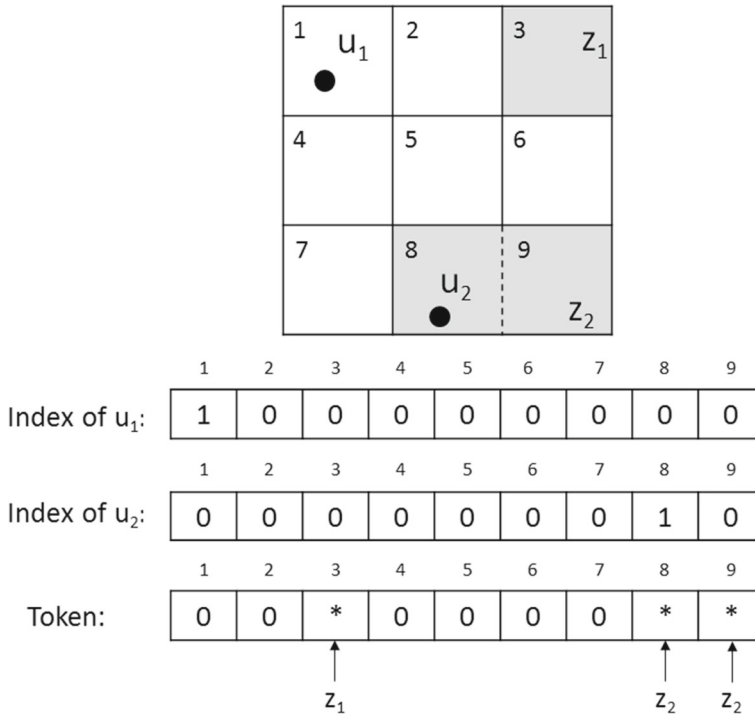
Fig. 4 Naive Baseline Encoding

evaluation for u_2 will return $Match$, as the position marked by '1' in the index of u_2 corresponds to a '*' in the token. Conversely, a $Non\text{-}Match$ is returned for u_1, as the bit '1' in position 1 corresponds to a '0' in the token. Algorithm 2 provides the baseline encoding pseudocode.


Algorithm 2 Baseline encoding.

---
1 p = ID of grid cell enclosing user u
2 A = set of grid cell IDs that make up the alert zone
   User u: Set $I = \begin{bmatrix} I_1 : I_{d^2} \end{bmatrix}$, $I_p = 1$ and $\forall i \neq p$, $I_i = 0$
   User u: Send encrypted I to Server S
   TA   : Set $T = \begin{bmatrix} T_1 : T_{d^2} \end{bmatrix}$, $T_i = *$ if $i \in A$ and $T_i = 0$ otherwise
   TA   : Send encrypted T to Server S
   Server: If $HVE\_Match(I, T) = True$, return $Match$
   Server: Otherwise, return $NotMatch$

As discussed in Appendix A, Eq. 1 from the query step executes two pairing operations and multiplies their results for every element in J, i.e., for every position that is not '*' in the token. Having a token with one position for each grid cell leads to high cost, so the naive encoding where the HVE width is equal to the number of cells is not practical. Furthermore, the sum of areas of all alert zones is relatively small compared to the entire dataspace, hence the number of '*' entries will be small, and the cardinality of set J will be large, increasing
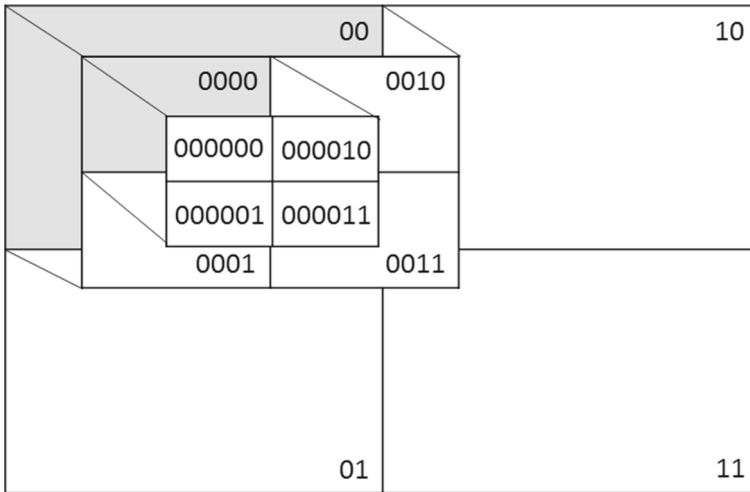
Fig. 5 Hierarchical partitioning on three levels

cost. Next, we propose two effective forms of encoding HVEs such that execution cost is reduced.

3.2 Hierarchical encoding

The main problem of the baseline encoding is that the HVE width grows linearly with the grid cell count. We propose a technique that reduces the HVE width from d^2 to $2\log d$, by using the binary representation of cell identifiers. However, the representation of the search predicates (and thus, that of the tokens) becomes more complicated, since the advantage of the "bitmap-like" representation of the baseline is lost. We investigate how to aggregate representations of adjacent cells belonging to the same alert zone, in order to reduce the amount of tokens required. Aggregation is performed according to a hierarchical spatial structure, hence the name of *hierarchical encoding*.

   We consider a logical organization of the grid cells into a quadtree-like structure[2] [35]. Figure 5 illustrates the space partitioning into four cells of equal size by using mediators on the Ox and Oy axes. Each of these four cells will have a 2-bit id: 00 for top left, 01 for bottom left, 10 for top right and 11 for bottom right. Next, each of these cells is partitioned recursively into four new cells, and the newly obtained 2-bit identifiers are concatenated as a suffix to the previous step identifiers. For simplicity, in this example we consider that the grid cell count is a power of 4, but any grid size can be accommodated in this model by using padding.

   The diagram also shows how aggregation of cells from level j is performed into a larger cell at level $j + 1$ (i.e., in reverse direction of scoping). Note that, with the binary representation of identifiers, cell aggregation corresponds to binary minimization of a logical 'OR' expression composed of the terms that represent cell identifiers. As a result, instead of using a distinct token (i.e., HVE pattern) for each cell, we can use token aggregation and reduce the number of predicates that need to be tested. If two cells are in the same alert zone and

---

[2]Note that this is a logical structure, no physical index is required.

their identifiers differ in just one bit, then a '*' can be used instead of that bit, similar to a wildcard in binary minimization. The newly obtained token is faster to generate and evaluate, because according to the operations described in Appendix A, only the positions in the pattern vector where the value is not '*' need to be considered (i.e., those in set J). If all of the four partitions belonging to the same quadtree node are in the same alert zone, then they can all be aggregated to the identifier of their parent. In our implementation, in order to generate HVE pattern vectors with aggregation, we use the binary expression minimization tool Espresso [32].

Consider for instance the example from Fig. 6, where the alert zone is composed of seven cells. All four cells whose identifiers have prefix 10 are in the zone, hence they can all be aggregated to $TK_1 = 10**$. Also, cells on the last vertical line can be aggregated to $TK_2 = 1*1*$. Finally, cells 0010 and 1010 can be aggregated to $TK_3 = *010$. Note that, although these tokens overlap, this does not introduce a correctness problem at query (i.e., matching) time at the server. Furthermore, the monitoring server can evaluate them in order from the most general (highest number of '*'s) to the most specific one (lowest number of '*'s). If one token evaluates to a $Match$ on a particular ciphertext, there is no need to evaluate the rest of the tokens, since it is clear that the user is in the alert zone. In addition, creating overlapping tokens helps if these tokens have more '*' symbols in their HVEs, because the cardinality of set J (Eq. 1 in Appendix A) decreases, hence query and token generation times decrease as well.

In summary, the hierarchical scheme works as follows:

Encryption. Users determine the binary identifier of the grid cell they are in, and create an HVE index I with that representation, having width $2 \log d$, where $d \times d$ is the grid
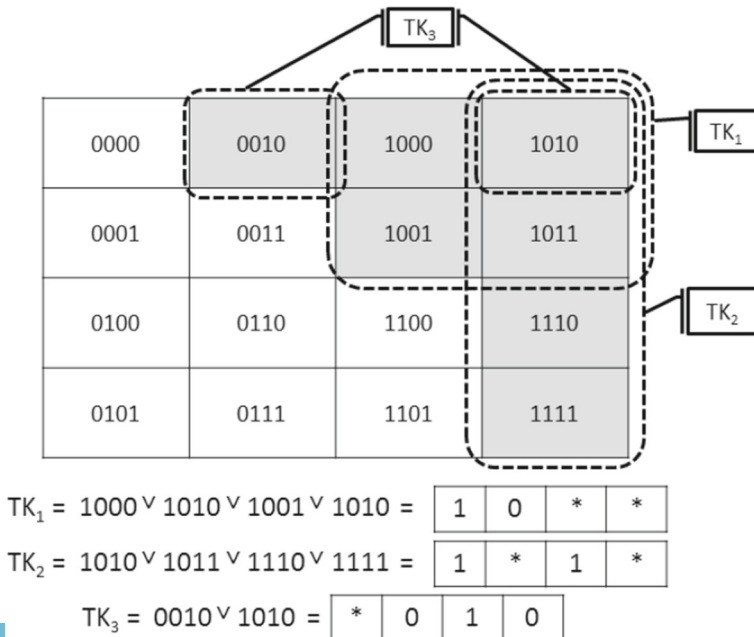

Fig. 6 Hierarchical encoding and token aggregation

size. The encryption is performed with respect to I. Since the grid parameters are public, the user can easily determine its enclosing cell and construct I.

Token Generation. For each alert zone z, the TA creates the set of binary codes of cells within the zone. Next, it computes the minimized binary expression equivalent to the logical 'OR' of all codes in the set. For each resulting term in the minimized expression, the TA creates a token, and the token will have a '*' symbol in each position that was reduced during the minimization. All tokens are sent to the server.

Query. For every user and alert zone, the server S performs matching as follows: S evaluates the encrypted user location against every token that represents the zone, in *decreasing* order of the number of '*' symbols in the token. In other words, tokens with a higher number of '*'s are considered first. If a $Match$ is obtained, then the remaining tokens for the zone are no longer considered. If a *Non-Match* is obtained for all tokens in the zone, then the server concludes that the user is not inside the zone.

Even though the number of tokens increases compared to the baseline, the width of each token is considerably smaller. In addition, the proportion of '*' symbols in a token is much higher for the hierarchical scheme, due to aggregation. Finally, considering tokens with a smaller J set first increases the chances of deciding on a $Match$ without having to consider all tokens of a zone. All these factors make the hierarchical encoding perform much faster than the baseline, as we show in Section 6. Algorithm 3 provides the hierarchical encoding pseudocode.

Algorithm 3 Hierarchical encoding.

---

1   p = ID of grid cell enclosing user u A = set of grid cell IDs that make up the alert zone

User u: Set $I = \left[I_1 : I_{2\log_2 d}\right]$, s.t. $\forall i, I_i \in \{0, 1\}$ and $\sum_{i=1}^{2\log_2 d} I_i \cdot 2^{i-1} = p$

User u: Send encrypted I to Server S

TA   : Create initial token set $TK = \left[TK_1 : TK_{2\log_2 d}\right]$, where $\forall a \in A, \exists T \in TK$
        s.t. $\forall i, T_i \in \{0, 1\}$ and $\sum_{i=1}^{2\log_2 d} T_i \cdot 2^{i-1} = a$

TA   : Reduce TK size by aggregating tokens within Hamming distance of 1

TA   : Send encrypted TK to Server S

Server: If $\exists T \in TK$ s.t. $HVE\_Match(I, T) = True$, return $Match$; Else return
        $NotMatch$

3.3 Gray encoding

The performance gain of the hierarchical technique comes from the ability to combine adjacent cells into a single search token with many '*' positions. In other words, the performance improves when the binary minimization of the logical 'OR' of cell identifiers is more effective. However, in some cases, no aggregation can be performed between two neighboring cells, as the Hamming distance between their identifiers is more than 1. As alert zones are composed of groups of neighboring cells, it is desirable to have small Hamming distance between adjacent cell identifiers. To improve the effectiveness of the binary minimization step, hence to increase the number of '*' values in search tokens, we represent cell identifiers using Gray codes [1]. This way, cell identifier values are assigned in such a manner that the Hamming distance between two adjacent cells is always 1, hence binary minimization is facilitated.

A one-dimensional Gray code vector is determined using the following recursive algorithm, where | represents the concatenation operator, and G_k is the vector of a Gray code instance at step k).

$$G_1 = (0, 1)$$

$$G_k = (g_1, g_2, ...g_{2^k})$$

$$G_{k+1} = (0|g_1, 0|g_2, ..., 0|g_{2^k}, 1|g_{2^k}, ..., 1|g_2, 1|g_1)$$

For $k = 3$, the following Gray code vectors are obtained:

$$G_1 = (0, 1)$$

$$G_2 = (00, 01, 11, 10)$$

$$G_3 = (000, 001, 011, 010, 110, 111, 101, 100)$$

Given a $d \times d$ grid, the length of the required Gray code necessary to represent all cells is $2^{\lceil log_2 d \rceil}$. We employ a Gray code instance independently for each of the two dimensions of the space, thus the identifier of a cell consist of the concatenation between the Gray code value for the y axis and the x axis values. Similar to hierarchical encoding, the scheme assumes a total number of cells that is a power of 4, but other cases can be handled by padding.

Figure 7 illustrates the advantage of using Gray encoding instead of hierarchical encoding for a 16-cell 4×4 grid. The alert zone consists of four cells. The two digits leading each row in the Gray encoding diagram (Fig. 7a) mark the two-bit prefix shared by all the cell identifiers in that row. Conversely, the two digits on top of each column mark the two-bit suffix of all the cell identifiers in that column. Using binary minimization, the two tokens shown in the diagram are obtained, each of them having one '*' symbol. Figure 7b shows how the hierarchical encoding behaves for the same input. Due to the fact that moving from cell 1001 to 1100, or from 0011 to 1001 corresponds to a Hamming distance larger than 1, no aggregation is possible between these cell pairs. As a result, three tokens are necessary to represent this zone. Furthermore, two of these tokens have no '*' symbol, leading to more expensive evaluation. As we will show in Section 6, Gray encoding achieves more effective aggregation, especially in the case of skewed data (e.g., Gaussian distribution of alert zones).
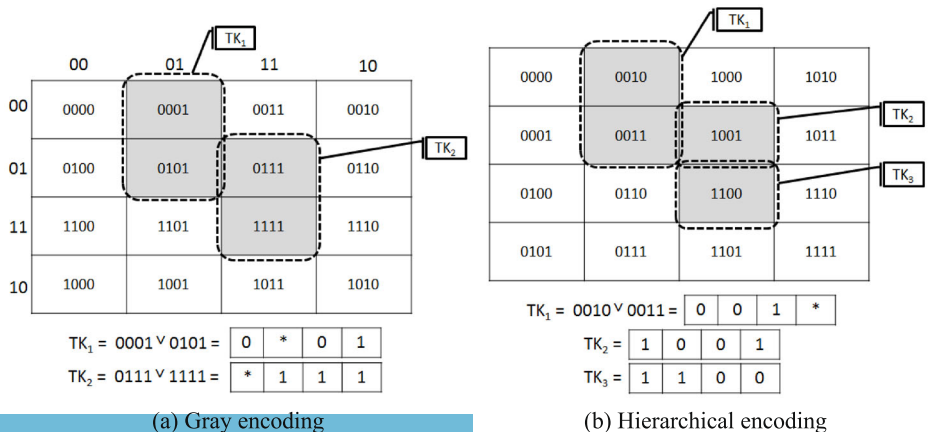

Fig. 7 Token Aggregation: Gray vs hierarchical encoding

The phases of encryption and query for the Gray encoding method are similar to their counterparts for the hierarchical encoding method of Section 3.2. The main difference is in the token generation phase, where the binary minimization is performed according to the Gray code cell identifier binary representation. As we show in the experimental evaluation (Section 6), using the Gray code representation can improve performance by achieving more effective binary minimization. This in turn results in either fewer tokens, or tokens with a larger proportion of '*' symbols.

4 Performance optimizations

4.1 Preprocessing mathematical operations

As discussed in Appendix A, the HVE mechanism involves a large number of exponentiations with very large integers which incur a significant computational cost. Fortunately, many of these exponentiations are performed on a common base. For example, if we take into consideration the encryption phase, in order to compute C' and C_0, A and V must be raised to the power of s. Even if s is chosen randomly for each run of this step, A and V depend on the public key, which remains unchanged for long periods of time (in commercial systems, re-keying can be done with frequency of once per year, or even less often). Furthermore, when computing $C_{i,1}$, because the index attributes consist of a vector of 0 and 1 values, the base of the power s can have only two values: H_i or $U_i \times H_i$. Because both depend only on the public key, these two exponentiations will always have a constant base. The same logic can be also applied to the exponentiations for token generation. By employing preprocessing on each of these fixed bases, the exponentiations become a lot faster. The preprocessing can be done offline, and the results used during online operation, leading to significant execution time savings.

When matching a token against an encrypted message, several pairing computations are performed. For a particular token, the values of K_0, $K_{i,1}$ and $K_{i,2}$ remain constant. When applying a pairing, Miller's algorithm is used [33]. Typically, for each such operation, it is required to compute several line equations. In [31] it is shown that effective preprocessing can be used as long as the first parameter is constant because the equations of the lines can be calculated and stored ahead of time. At runtime, the coordinates of a given point are substituted into these precomputed expressions. Since HVE requires symmetric elliptic curves, preprocessing can be also done for the second parameter. Preprocessed information is stored with each token and used by the server to improve the time of each pairing. Preprocessing each token must be done only when the tokens are generated at the trusted authority.

4.2 Parallelization

The server is monitoring a large number of users, and may receive a large number of alert zones. This creates a considerable load on the server. However, we emphasize that the processing of a message from a user can be done independently from messages originating at other users. Furthermore, even for the same user, the matching for different alert zones are completely independent operations. This presents a great potential for parallelization. In fact, the problem is embarrassingly parallel, and significant execution time improvements can be obtained by using several CPUs for matching. Nowadays, even off-the-shelf desktop computers have multiple cores. Commercial cloud services typically have hundreds or thousands of CPUs available for computation. Due to the parallel nature of the problem, the

speedup is expected to be close to linear, and the resulting system scales very well as the number of CPUs involved grows.

We consider a message-passing parallel computing paradigm, which is favored by the fact that only a small amount of data needs to be shared among distinct CPUs.

One master process coordinates all other slave processes. The master process distributes to the slaves the search tokens. Then, as encrypted updates from users arrive, the master receives the requests and dispatches them to slaves. Load balancing can be easily implemented at the master level, which keeps track of the status of all slave processes. No communication is required between slave processes, and the master-slave communication is required only at the start and end of each task. Furthermore, distinct messages originating from the same user can be processed on different CPUs without any loss in correctness or performance (i.e., no state maintenance is required). After processing is done, if the token evaluated successfully, a response action can be taken by the processing CPU, or the event can be sent to a central server responsible only with handling what to do in case of a successful match.

5 Privacy-performance trade-off through alert zone expansion

So far, we considered that alert zones are a fixed input to the system, and we provided data encodings and optimizations to reduce computational overhead under this constraint. Since alert zones were not modified, we maintained the amount of location disclosure to a minimum, i.e., an adversary could only learn whether a specific ciphertext corresponded to a location inside the alert zone or not. In this section, we consider a relaxation of the alert zone extent in order to improve performance. Specifically, given an input alert zone, we investigate whether it is possible to slightly enlarge it such that the resulting set of tokens needed to implement secure notification requires fewer bilinear pairings to evaluate.

To maintain the level of additional disclosure low, we allow only a relatively small enlargement factor, expressed as a ratio of the alert zone area, and quantified by a bound parameter α. Given an enlargement factor α, our proposed alert zone expansion heuristic determines an enlarged area with significantly lower processing overhead. In effect, this proposed optimization trades a small amount of additional location disclosure for a significant boost in matching performance. As a salient feature of this optimization, the privacy-performance trade-off can be tuned using a single parameter (α).

The optimization is deployed at the TA, which is in charge of generating search tokens. In an actual deployment, since the TA is trusted, it can perform additional steps to check whether the enlarged zone is acceptable from a security standpoint, for instance by comparing it against a set of pre-defined policies. In this paper, we only focus on the performance aspect, and derive effective algorithms that quickly generate enlarged search tokens (the policy aspect is outside our scope). Similar to optimizations from prior sections, the zone expansion is guided by the objective of deriving tokens with fewer non-wildcard elements, which results in less computation. The expansion technique assumes the same hierarchical data domain representation considered so far, and works in conjunction with either hierarchical or Gray encodings.

We denote by *base cell* a cell in the leaf level of the hierarchical domain representation (recall that, the domain is split into $d \times d$ base cells, where d is a power of two). The hierarchy has a number of $1 + \log d$ levels. At level k, an *aggregate* cell consists of $2^k \times 2^k$ base cells. Specifically, at the leaf level, numbered as $k = 0$, each cell is a base cell, whereas at the top of the hierarchy (level $\log d$) there is a single cell with size $d \times d$ (expressed in

terms of base cells). We identify a cell at level k by its coordinates within that level: $(x, y)_k$. The *binary identifier* of a cell consists of a binary string, which can be immediately derived from its coordinates.

Algorithm 4 captures the main steps of the proposed heuristic alert zone expansion technique. The input consists of expansion factor α and initial alert zone A. The heuristic is given a maximum budget $W = \lfloor \alpha |A| \rfloor$ base cells that it can add to the initial zone, where $|A|$ is the area of the initial zone expressed in terms of base grid cells. The output of Algorithm 4 is an expanded zone $\hat{A}$ such that $|\hat{A}| \leq |A| + W$ and the number of bilinear pairings required to evaluate $\hat{A}$ is lower than that of A.

Algorithm 4 ExpandZone.

Input: expansion ratio α; alert zone A
Output: expanded alert zone $\hat{A}$

1   $W \leftarrow \lfloor \alpha |A| \rfloor, \mathcal{B} \leftarrow \emptyset$
2   $\hat{A} \leftarrow CrtZone \leftarrow A$
3   **for** $k \in [0, \ldots, \log d]$ **do**
4      $PSet \leftarrow$ SelectPatchesSingleLevel$(W, \frac{d}{2^k}, CrtZone)$
5   **for** $p \in PSet$ **do**
6   **for** $(x, y)_k \in p.attached\_cells$ **do**
7            $\mathcal{B} \leftarrow \mathcal{B} \cup$ all base cells belonging to cell $(x, y)_k$
8   **if** *#pairing of* $(\hat{A} \cup \mathcal{B}) \leq$ *#pairings of* $\hat{A}$ **then**
9         $\hat{A} \leftarrow \hat{A} \cup \mathcal{B}$
10   **for** $p \in PSet$ **do**
11           $W \leftarrow W - p.cost$
12   **for** $(x, y)_k \in p.attached\_cells$ **do**
13              $CrtZone \leftarrow CrtZone \cup (x, y)_k$
14        $W \leftarrow \lfloor \frac{W}{4} \rfloor$
15   **if** $W \leq 0$ **then**
16   **break**
17   **for** $(x, y)_k \in CrtZone$ **do**
18           $CrtZone = CrtZone \setminus (x, y)_k \cup (\lfloor \frac{x}{2} \rfloor, \lfloor \frac{y}{2} \rfloor)_{k+1}$
19 **return** $\hat{A}$

The ExpandZone routine (Algorithm 4) works by considering each level of the data domain hierarchy.

An essential step of ExpandQuery is the SelectPatchesSingleLevel routine (detailed in Algorithm 9) which finds *patches* to add to the current set of zone cells (line 4). A *patch* (formally defined in Section 5.1) is a set of cells added to the zone in a single iteration. If the new set of zone cells, denoted as $CrtZone$, does not require more pairings than the current set of cells, an expansion is made with the cells in the patch and we continue to the next level.

In order to prepare for the next level (lines [9-18]), parameters and indices are adjusted. Budget W is reduced by a factor of 4, since in the next level the size of one cell is equal to 4 cells in the current level. Similarly, indices of zone cells are divided by two. The intuition behind dividing the indices by two is that all 2×2 areas containing zone cells in this level must be fully covered by the expansion, which means all cells in those areas are in $CrtZone$.

(a) Level $k = 0$ (b) Level $k = 1$
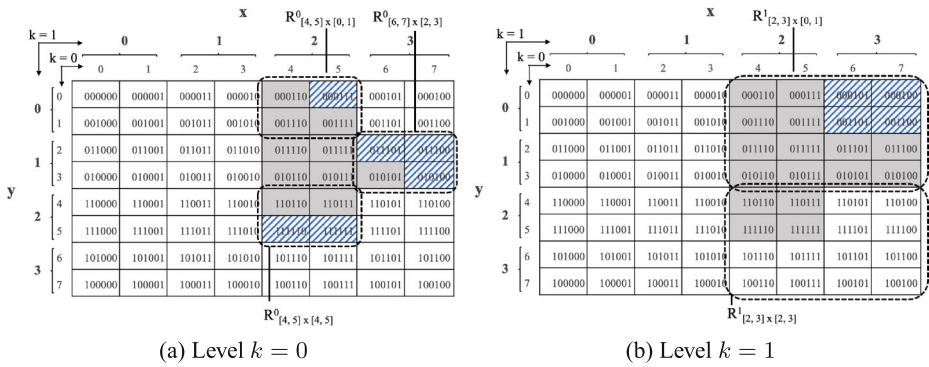
Fig. 8 Example of expanding alert zone (in grey) at level $k = 0$ (**a**) and $k = 1$ (**b**). At level 0, a total of 6 cells (illustrated with stripes) are added to obtain 3 fully-covered areas $R^0_{[4,5]\times[0,1]}$, $R^0_{[6,7]\times[2,3]}$, and $R^0_{[4,5]\times[4,5]}$. At level 1, four additional cells are added to the area $R^1_{[2,3]\times[0,1]}$. The final expanded alert zone contains all cells in $R^0_{[4,7]\times[0,3]}$ and $R^0_{[4,5]\times[4,5]}$ (both grey and diagonally-striped cells in (**b**))

Algorithm 4 stops when one of the following conditions is met: *(i)* the new set of zone cells increases the number of pairings; *(ii)* budget W is exhausted; or *(iii)* the zone expands to the entire root level.

To illustrate the zone expansion algorithm, consider the example in Fig. 8. Zone cells are shown in grey color, and budget is set to $W = 10$. An area with $x \in [x_1, x_2]$, $y \in [y_1, y_2]$ at level k is denoted as $R^k_{[x_1,x_2]\times[y_1,y_2]}$. Starting at level $k = 0$ (Fig. 8a), the 2×2 areas $R^0_{[4,5]\times[0,1]}$, $R^0_{[6,7]\times[2,3]}$, and $R^0_{[4,5]\times[4,5]}$ are considered for expansion. All six cells with diagonal stripes are added to the current set of zone cells to fill those three areas. To prepare for expansion at the next level, the coordinate ids of zone cells must be adjusted for each 2×2 area. For example, for $k = 1$, area $R^0_{[4,5]\times[0,1]}$ becomes cell $(2, 0)_1$, area $R^0_{[6,7]\times[2,3]}$ becomes cell $(3, 1)_1$ and so on. The budget W is reduced to $\lfloor \frac{10-6}{4} \rfloor = 1$. Next, at level $k = 1$ (Fig. 8b), the areas $R^1_{[2,3]\times[0,1]}$ and $R^1_{[2,3]\times[2,3]}$ are considered for expansion. The cells with diagonal stripes in range $R^1_{[3,3]\times[0,0]}$ (which equals $R^0_{[6,7]\times[0,1]}$ in the base grid) are added to the zone.

5.1 Patch assembly

Next, we focus on the process of assembling *patches* at each level k of the data domain hierarchy.

A patch is a set of cells that can be combined with existing zone cells to reduce the number of non-wildcard elements in a search token.

We denote the cells belonging to a patch as *attached cells*, and the zone cells adjacent to the patch as *attaching cells*. A patch is associated with a local *cost* and *gain*: the cost measures the increase in alert zone area, whereas the gain quantifies the resulting reduction in bilinear pairing operations when the patch is added to the zone.

We consider as patch candidate each 2×2 cell[3] $R^k_{[x,x+1]\times[y,y+1]}$ that satisfies the following conditions: *(i)* has even x and y coordinates, *(ii)* contains at least one zone cell, and

---

[3]We emphasize that, as patch candidates are considered at each level of the hierarchy, a patch cell may include many base grid cells.

(iii) has at least one non-zone cell. Revisiting the example in Fig. 8a, the area $R^0_{[4,5]\times[0,1]}$ composed of 2×2 base cells is a patch candidate. Note that, not all 2×2 cell areas are valid candidates for patches. For instance, $R^0_{[5,6]\times[0,1]}$ has an odd x; $R^0_{[6,7]\times[0,1]}$ does not contain any zone cell; and $R^0_{[4,5]\times[2,3]}$ does not contain any non-zone cell.

For each valid patch candidate $R^k_{[x,x+1]\times[y,y+1]}$, cells are indexed in a spiral order, as shown in Fig. 9a. We use this indexing order because it simplifies the process of patch assembly, as will be described later in Section 5.2. In order to keep track of zone and non-zone cells, a boolean array *marked* is maintained, such that *marked*[i] = *True* if i^{th} cell within $R^k_{[x,x+1]\times[y,y+1]}$ is a zone cell, and *marked*[i] = *False*, otherwise. Figure 9b shows a *marked* array for the area in Fig. 9a. The *marked* array is constructed by checking for each cell within the area whether or not it belongs to the alert zone. The marking procedure is summarized in Algorithm 5.

Using the *marked* array, patch candidates are constructed such that one or more non-zone cells can be attached to zone cells to reduce the number of pairings. Figure 10 shows several examples of patches for an area with 2×2 cells containing one, two, or three zone cells. In each example, the non-zone cell (striped fill) is attached to the zone cell (grey fill) to form a patch. Note that in Fig. 10c, a striped cell can be attached to either grey cell.

Algorithm 5 MarkZoneCells.

Input: zone cells A; start values x and y of area $R^k_{[x,x+1]\times[y,y+1]}$
Output: boolean array indicating which cells in 2×2 block are zone cells
1 Initiate *marked* array with *marked*[i] = *False*, $\forall i \in [0, 3]$
2 **foreach** $i \in [0, 3]$ **do**
3   **if** i^{th} cell of $R^k_{[x,x+1]\times[y,y+1]} \in A$ **then**
4   *marked*[i] $\leftarrow True$
5 **return** *marked*

However, when the area contains only one zone cell, although there are three potential patches, only one of these is selected (Fig. 10a). The reason is that if two patches, each having a single zone cell, are selected, the number of pairings is not reduced; on the other hand, if the patch with three zone cells is selected, there is no need to select other patches with a single zone cell. Therefore, for each area, we construct *patch groups* that include all potential patches such that no more than one patch can be selected from that group. For

(a)   (b)
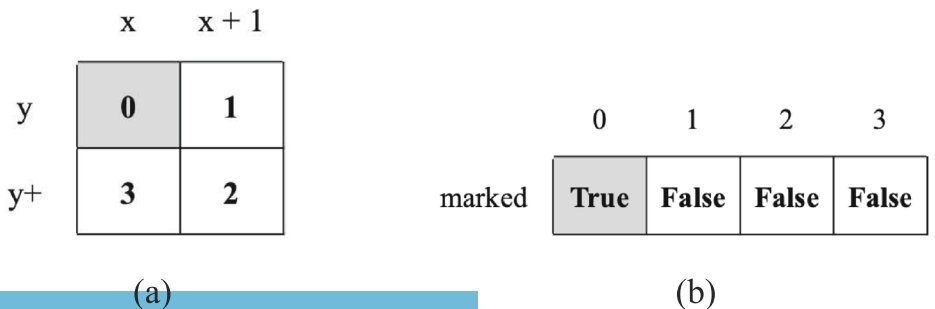
Fig. 9 **a** Cell indices within a 2×2 patch; **b** Array *marked* for a 2×2 patch

example, in Fig. 10a, there is only one patch group containing all three patches; in Fig. 10b and d, there is only one patch group containing one patch; in Fig. 10c, there are two patch groups, each containing one patch.

The GetPatchGroupsInsideArea routine (Algorithm 6) shows the details of constructing patches and patch groups. The algorithm handles separately each case based on the number of zone cells in the area. For a single zone cell (line 3), similar to the example in Fig. 10a, one patch group is constructed which includes two patches: one with one non-zone cell and another with all three non-zone cells. If there are two zone cells (line 7), the algorithm further considers if those two zone cells are adjacent or opposite (similar to Fig. 10b and c, respectively) and either one or two patch groups are created, corresponding to the two situations. Finally, when there are three zone cells (line 17), a single patch group is created.


Algorithm 6 GetPatchGroupsInsideArea.

Input: array *marked* for 2×2 cell block
Output: Patch groups with cell ids 0, 1, 2, or 3

1 $\mathcal{G}_{inside} \leftarrow \emptyset$
2 *num_zone_cells* $\leftarrow$ number of 1's in *marked*
3 **if** *num_zone_cells* $= 1$ **then**
4      $p_1 \leftarrow$ new patch with an non-zone cell adjacent to the zone cell as *attached_cells* and the zone cell as *attaching_cells*
5      $p_2 \leftarrow$ new patch with all three non-zone cells as *attached_cells* and the zone cell as *attaching_cells*
6      $\mathcal{G}_{inside} \leftarrow \{p_1, p_2\}$
7 **else if** *num_zone_cells* $= 2$ **then**
8   **if** *two zone cells are adjacent* **then**
9          $p \leftarrow$ new patch with all non-zone cells as *attached_cells* and the zone cells as *attaching_cells*
10         $\mathcal{G}_{inside} \leftarrow \{p\}$
11   **else**
12         $p_1 \leftarrow$ new patch with an non-zone cell adjacent to the first zone cell as *attached_cells*
13         $p_2 \leftarrow$ new patch with the other non-zone cell adjacent to the second zone cell as *attached_cells*
14         $G_1 \leftarrow \{p_1\}$
15         $G_2 \leftarrow \{p_2\}$
16         $\mathcal{G}_{inside} \leftarrow \{G_1, G_2\}$
17 **else if** *num_zone_cells* $= 3$ **then**
18     $p \leftarrow$ new patch with all non-zone cells as *attached_cells* and the zone cells as *attaching_cells*
19     $\mathcal{G}_{inside} \leftarrow \{p\}$
20 **return** $\mathcal{G}_{inside}$

At the end of Algorithm 6, each patch has its cells numbered from set $\{0, 1, 2, 3\}$. In order to recover the original cell ids (i.e., the coordinates in current level k of hierarchy), we use Algorithm 7, which takes as inputs a cell id $i \in [0, 3]$ and the x, y values of the area $R^k_{[x,x+1] \times [y,y+1]}$, and utilizes the spiral index to recover the original values.
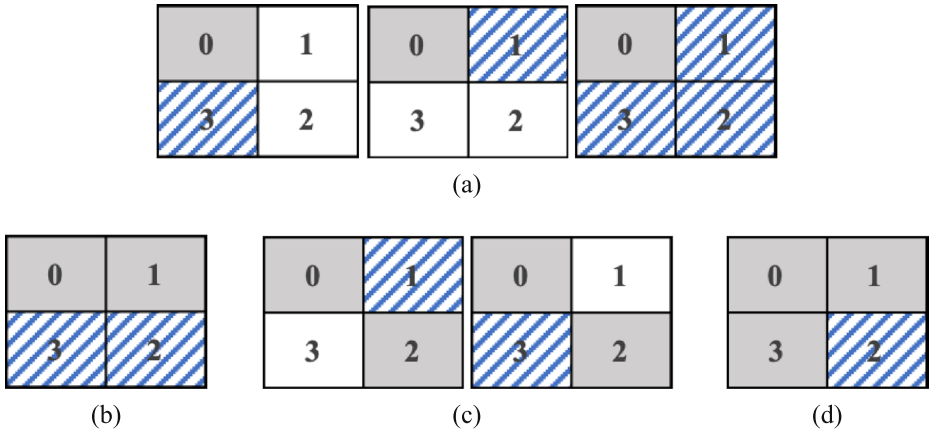
Fig. 10 Example of candidate patches for an area of size 2×2 **a** Three candidate patches for an area with one zone cell; **b** One candidate patch for an area with two zone cells; **c** Two candidate patches for an area with two zone cells; **d** One candidate patch for an area with three zone cells

---

Algorithm 7 RecoverOriginalCoordId.

Input: cell id $i \in [0, 3]$; x, y values of the area $R^k_{[x,x+1] \times [y,y+1]}$
Output: Original cell coordinates
1 **if** $i = 1$ **or** $i = 2$ **then**
2     $x \leftarrow x + 1$
3 **if** $i = 2$ **or** $i = 3$ **then**
4     $y \leftarrow y + 1$
5 **return** (x, y)

Next, we need to evaluate which patches are more desirable to use in the enlarged zone, by computing the local *cost* and *gain* for each patch. Algorithm 8 takes as inputs a candidate patch and the grid dimension d_k at current level k. It outputs as cost the number of attached cells (i.e., non-zone cells) of the patch (line 3). Effectively, the cost measures the amount of enlargement of the expanded alert zone caused by this patch. The gain measures the amount of saved computation: specifically, the number of search token non-wildcards that are eliminated when we combine the attached cells with the attaching cells for the current

Table 1 Example of candidate patch groups for expanding the alert zone at level $k = 0$

Patch group	Patch	cost	gain	attached_cells	attaching_cells
G_1	p_1	1	6	$(5, 0)$	$(4, 0), (4, 1), (5, 1)$
G_2	p_2	1	1	$(6, 2)$	$(6, 3)$
	p_3	1	1	$(7, 3)$	$(6, 3)$
	p_4	3	2	$(7, 2), (7, 3), (6, 2)$	$(6, 3)$
G_3	p_5	2	1	$(4, 5), (5, 5)$	$(4, 4), (5, 4)$

Patch groups G_1, G_3 contain one patch, while patch group G_2 contains 3 patches. No more than one patch is selected per patch group

patch. There are two cases to consider when determining the gain of the patch: *(i)* when only one cell is attached to form a 1×2 patch (line 4), we can remove one non-wildcard element (line 5); *(ii)* when the entire 2×2 area is filled (line 6), the number of zone cells inside the area (i.e., n_1) is further considered to determine the gain. Specifically, when $n_1 = 3$, the gain is larger ($2 \times k$) since we can remove a token in its entirety.

Algorithm 8: CalculateCostGain.

Input: patch p; grid length at current level k;
Output: updated patch with cost and gain calculated

1   $n_1 \leftarrow$ number of zone cells of p
2   $n_2 \leftarrow$ number of non-zone cells of p
3   $p.cost \leftarrow n_2$
4   **if** $n_1 + n_2 = 2$ **then**
5      $p.gain \leftarrow 1$
6   **else if** $n_1 + n_2 = 4$ **then**
7   **if** $n_1 = 1$ **then**
8         $p.gain \leftarrow 2$
9   **else if** $n_1 = 2$ **then**
10       $p.gain \leftarrow 1$
11   **else**
12       $p.gain \leftarrow 2 \times k$
13   **return** p

In the previous example from Fig. 8a, there are three patch groups corresponding to three areas: G_1 for area $R^0_{[4,5] \times [0,1]}$, G_2 for area $R^0_{[6,7] \times [2,3]}$, and G_3 for area $R^0_{[4,5] \times [4,5]}$. The patches in each patch group along with their cost, gain, attaching cells, and attached cells are shown in Table 1. For instance, to express area $R^0_{[4,5] \times [0,1]}$ one can look at patches p_1 of G_1, and use two tokens "00*110" and "00111*", with a total of 10 non-wildcard elements. By adding one cell, only a single token "00*11*" is needed to represent the area. Thus, the number of non-wildcard elements is reduced from 10 to 4, or an improvement of 6. The high gain when applying patch p_1 results not only from the number of non-wildcards reduced in one token, but also from the reduction in the number of tokens (as one of the initial tokens is completely eliminated).

5.2 Patch selection

Once we have the set of patches and patch groups, as well as their respective costs and gains, we need a method to select the actual patches to expand the current alert zone. Algorithm 9 outlines the patch selection process, which takes as inputs budget W, the grid dimension at current level $d_k = \frac{d}{2^k}$, and current alert zone A_k. It outputs a set of patches *PSet* that has total cost at most W and maximizes the gain compared to other candidate patches.

The selection algorithm works within an expanding search boundary determined by the call to routine `FindExpandingBoundary` in line 1 (`FindExpandingBoundary` is summarized in Algorithm 10: the boundaries consist of the maximum and minimum coordinate values of zone cells, and they always have even values). Then, for each 2×2 area starting with even values (lines [3–6] in Algorithm 9), if the current 2×2 area is a valid area to expand (line 8), the patches and patch groups within this area are constructed (according to the procedure detailed in Section 5.1). First, the cells that already belong to the current

zone are marked by calling Algorithm 5 (line 9). Then, using the marking information, the set $\mathcal{G}_{inside}$ of *patch groups* within that area is constructed by calling Algorithm 6 (line 10). Next, for each patch in the patch groups of $\mathcal{G}_{inside}$, the original coordinate ids of cells in the attaching and attached set of that patch are recovered by calling Algorithm 7 (line 13), and the local *cost* and *gain* are calculated by calling Algorithm 8 (line 14). Finally, a set of patches *PSet* is selected for expansion by calling Algorithm 11 (line 16).

Algorithm 9 SelectPatchesSingleLevel.

Input: budget W; grid dimension d_k at level k; zone cells A_k
Output: patches with total cost $\leq W$ and positive gain

1 $min\_x, max\_x, min\_y, max\_y \leftarrow$ FindExpandingBoundary(d_k, A_k)
2 $\mathcal{G} \leftarrow \emptyset$
3 $x \leftarrow min\_x$
4 **while** $x < max\_x$ **do**
5      $y \leftarrow min\_y$
6   **while** $y < max\_y$ **do**
7          $c \leftarrow$ number of zone cells in this area $R^k_{[x,x+1] \times [y,y+1]}$
8   **if** $0 < c < 4$ **then**
9              $marked \leftarrow$ MarkQueryCells(A_k, x, y)
10              $\mathcal{G}_{inside} \leftarrow$ GetPatchGroupsInsideArea($marked$)
11   **for** $G \in \mathcal{G}_{inside}$ **do**
12   **for** $p \in G$ **do**
13   Recover original coordinate ids of each cell i in *attaching_cells* and *attached_cells* using RecoverOriginalCoordId(i, x, y)
14   Update $p \leftarrow$ CalculateCostGain(p, d_k)
15                  $\mathcal{G} \leftarrow \mathcal{G} \cup G$
16 $PSet \leftarrow$ KnapsackForGroups($W, \mathcal{G}$)
17 **return** *PSet*

Algorithm 10 FindExpandingBoundary.

Input: Grid length d_k; zone cells A_k
Output: the boundary for expansion

1 ($min\_x, max\_x, min\_y, max\_y$) = minimum bounding rectangle of A_k expanded to even coordinates
2 **return** $min\_x, max\_x, min\_y, max\_y$

The patches are selected such that the total cost is no more than W, the total gain is maximized, and there is no more than one patch selected per group. This can be modeled as a variant of a multiple-choice knapsack problem (MCKP) where a class in MCKP is represented by a patch group, and we may choose a single item from a class, instead of being required to choose at least one item. The reduction is as follows: Given an instance of MCKP with capacity W, m classes, and each item j in class j having cost $c_{i,j}$ and gain $g_{i,j}$, for each class i, a new item j' (or patch in our setting) is added with cost $c_{i,j'} = 0$ and gain $g_{i,j'} = 0$. However, our patch selection problem is not NP-hard, because W is restricted to a fraction of the alert zone, which in turn is restricted to a fraction of the entire grid.

In our setting, each patch group contain either one or two patches. As a result, a dynamic programming approach for traditional binary knapsack problem can be used. Algorithm 11 shows the dynamic programming solution that returns the selected patches for expansion. In the example summarized in Fig. 8 and Table 1, patches p_1, p_4, p_5 are selected for expansion at level $k = 0$.

Algorithm 11: KnapsackForGroups.

Input: capacity W; groups of N_g patches $\mathcal{G} = \{G_1, G_2, \ldots, G_{N_g}\}$;
Output: patches with total cost $\leq W$, total gain is maximized, and each patch belongs
        to a different group

1 $K \leftarrow$ matrix of size $(W + 1) \times (N_g + 1)$
2 **for** $i \in [0, N_g]$ **do**
3   **for** $w \in [0, W]$ **do**
4   **if** $i = 0$ **or** $w = 0$ **then**
5             $K[i][w] \leftarrow 0$
6   **else**
7             $K[i][w] \leftarrow K[i - 1][w]$
8   **for** $p \in G_i$ **do**
9   **if** $p.cost \leq w$ **then**
10                    $K[i][w] \leftarrow \max(K[i][w], p.gain + K[i - 1][w - p.cost])$
11 $PSet \leftarrow \emptyset$
12 $i \leftarrow N_g$
13 $w \leftarrow W$
14 **while** $0 < K[i][w]$ **do**
15   **for** $p \in G_i$ **do**
16   **if** $p.cost \leq w$ **and** $K[i][w] = p.gain + K[i - 1][w - p.cost]$ **then**
17             $PSet \leftarrow PSet \cup \{p\}$
18 $w \leftarrow w - p.cost$
19 **return** $PSet$

5.3 Complexity analysis

The complexity of the alert zone expansion (Algorithm 4) depends on the complexity of the binary minimization step (line 7) in which the algorithm decides whether or not to continue expansion. In the worst case, Algorithm 4 needs to expand through all $\log d$ levels of the hierarchy, and in each level its invokes Algorithm 9 and the binary minimization procedure (in our implementation, we use the Espresso tool [32]).

Since Algorithm 9 finds the patch groups within the boundary of the query, and the size of the alert zone is often much smaller than the size of the data domain, we formulate the complexity of Algorithm 9 based on the alert zone size. Let $P_k = |A_k|$ be the number of cells of the alert zone at level k. After finding patch groups, Algorithm 9 invokes the dynamic programming solution in Algorithm 11 to select patches. In the worst case, the number of patch groups N_g at level k equals the number of cells P_k of the alert zone. In our setting, there are only one or two patches in each patch group. Hence, the complexity of Algorithm 11 becomes $\mathcal{O}(N_g W) = \mathcal{O}(\alpha P_k^2)$ since $W = \alpha P_k$. Thus, the complexity of Algorithm 9 is $\mathcal{O}(P_k + \alpha P_k^2)$.

However, since the value of P_k is divided by a factor of 4 each time k increases, the complexity of the alert zone expansion (Algorithm 4) becomes $\mathcal{O}(P_0 + \alpha P_0^2 + T_{Es}((1 +$

(a) Token generation   (b) Message encryption   (c) Query
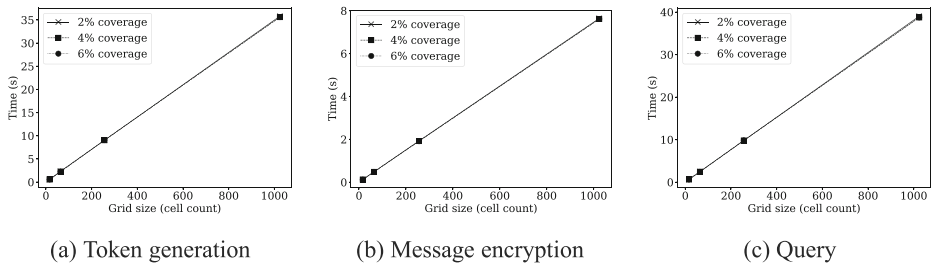
Fig. 11 Baseline encoding results

$\alpha) P_0) \log d)$ where P_0 is the size of the zone at the base level (i.e., original grid) and $T_{Es}(t)$ is the time to run the binary minimization procedure for t inputs.

6 Experimental evaluation

We implemented a Python prototype of the proposed HVE-based location-based alert system and performance optimizations. We have used as dataset the city of Oldenburg, and generated user movements using Brinkhoff's *IAPG Network-based Generator of Moving Objects*[4] [7]. We generated alert zones within the boundaries of the dataset domain according to two distributions: uniform and Gaussian. We vary the percentage of space covered by alert zones compared to the entire dataspace extent from 1% to 10%, and we denote this parameter as *coverage*. We consider a regular grid partitioning the two-dimensional space with size ranging from 16 to 1024. The HVE cryptographic functions were implemented using the Gnu MP v6.1.2 library and the Pairing-Based Cryptography v0.5.14 library.[5] We use key lengths of 768, 1024 (default value), 1280 and 1536 bits.

The experimental testbed consisted of a Intel(R) Core(TM) i9-9980XE CPU (3.00GHz) with 18 cores and 128GB of RAM, running Ubuntu 18.04 LTS. All code was written in Python 3.6.9.

6.1 Baseline evaluation

Figure 11 shows the execution time results obtained for token generation, encryption and query. The times presented are for a single operation, and present the average value obtained for a particular grid size and percentage of the area covered by alert zones (each percentage value has a different line in the graphs). First, we note that the coverage does not have a significant effect on the execution time, because the width of the HVE obtained is so large that the associated overhead overshadows the influence of the additional '*' symbols obtained as the area of alert zones grows. Second, it can be observed that the values obtained are very large, and clearly not acceptable in practice.

Token generation can take up to 35 seconds. Although expensive, it can be argued that the TA does not execute this phase very often (only when a new alert zone occurs), hence its performance is not critical. However, encryption is very frequent, and it is executed at the resource-constrained mobile users.

---

[4]http://iapg.jade-hs.de/personen/brinkhoff/generator/

[5]Available online at http://gmplib.org/ and http://crypto.stanford.edu/pbc/

It can take up to 8 seconds to generate a single encrypted update on a high-end CPU (in practice, this would be executed on a mobile phone). Furthermore, the time required at the server to process a single user update (i.e., perform matching against all alert zones) can reach 40 seconds.

6.2 Hierarchical and gray encoding

Figures 12 and 13 show the comparison results for uniform and Gaussian alert zone distributions, respectively. Hierarchical encoding clearly outperforms the baseline, especially in terms of encryption time. The maximum time required for encryption is less than 0.2 seconds, in contrast with 8 seconds for the baseline (Fig. 11b). Recall that alert zones do not influence encryption, so the hierarchical encoding lines present in Fig. 12b overlap. Encryption is also independent of alert zone distribution, so we do not show encryption in Fig. 13.

In terms of token generation and query time, the gain in performance is higher for the Gaussian distribution, since there is more potential for token aggregation. The reason is that minimization of binary expressions of cell identifiers is more effective when zones are clustered, which is likely to be the case in practice.

As expected, execution time is higher for finer-grained grids. However, as opposed to the baseline, in the case of hierarchical encoding the coverage has a significant effect on token generation and query performance, as more alert zone cells translate into a larger number of tokens. Still, the variation with coverage is sublinear, due to the good effectiveness of the aggregation strategy employed (note how when coverage doubles from 2% to 4% for uniform data and largest grid size, the query time increases only by 25%). Although the absolute execution times are still high, hierarchical encoding significantly outperforms the baseline. Later in Section 6.3 we show how optimizations can be used to further cut down the performance overhead. For the rest of the experimental evaluation, we will omit the baseline results.

Next, we evaluate the effect of using Gray encoding on performance. Recall from Section 3.3 that using Gray codes provides better potential for aggregation, thus reducing the number of required tokens and/or increasing the proportion of '*' symbols in a token. For uniform data (graph omitted due to space considerations), both encodings perform similarly, without a clear winner, due to the fact that the aggregation potential is equal in the two cases. On the other hand, for Gaussian data (Fig. 14) where alert zones are clustered, Gray encoding favors aggregation of cells. For clarity, to keep the number of lines in the graph low, we present the ratio between the execution time of Gray divided by that of hierarchical encoding. Lower values of the ratio correspond to higher gains for the Gray encoding.
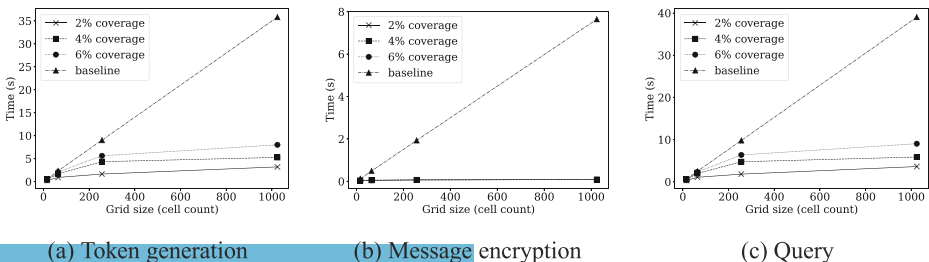

(a) Token generation   (b) Message encryption   (c) Query

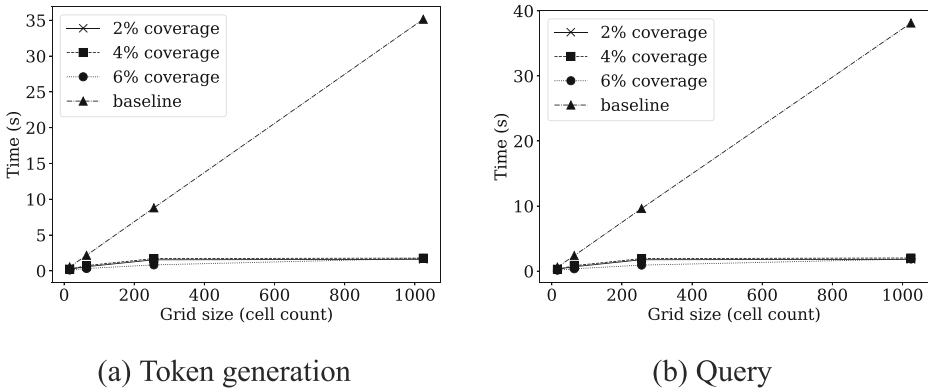**Fig. 12** Hierarchical encoding results on uniform data

Fig. 13 Hierarchical encoding results on Gaussian data

In practice, as alert zones are likely to be clustered, Gray can bring significant performance benefits, of up to 60%.

6.3 Optimization effect

We evaluate the performance of the proposed techniques when incorporating the performance optimizations discussed in Section 4.

First, we show the effect of incorporating preprocessing to pre-compute and re-use some of the results to expensive mathematical operations, such as exponentiations with large numbers.

Figure 15 present the absolute token generation and query times for both proposed encoding techniques on uniform data and two values of the alert zone coverage, namely 2% and 4% (Gaussian data results show similar trends, so we omit them for brevity). Token generation computation requirements are improved by roughly $25 - 30\%$.

As the coverage increases, more tokens are required to represent a zone, so the generation time increases. We believe that such times are reaso nable in practice, especially since
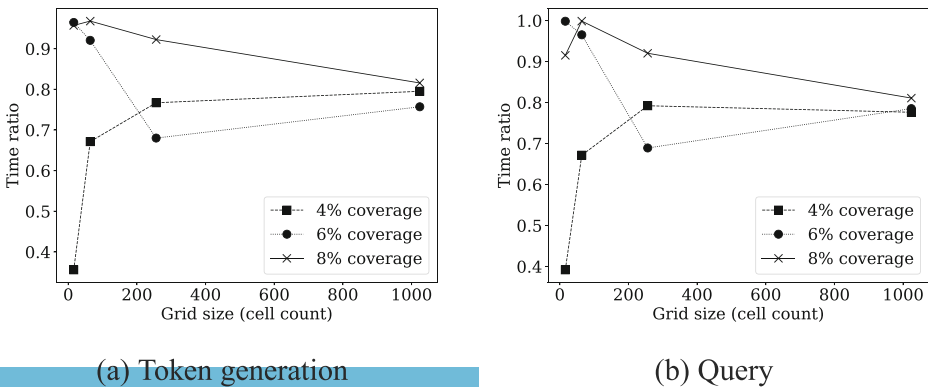
Fig. 14 Gray vs. hierarchical encoding on Gaussian data

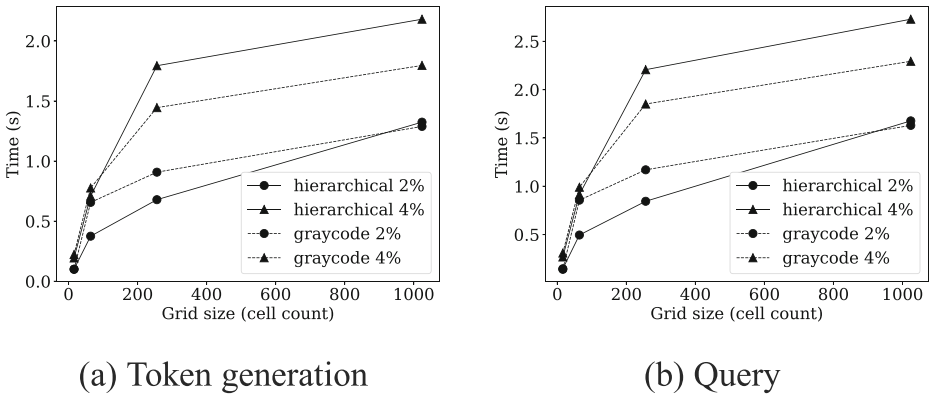(a) Token generation   (b) Query

Fig. 15 Effect of preprocessing on uniform data

creation of alert zones is not a frequent event in the system operation. In terms of query-ing, the execution times are approximately cut in half compared to the non-optimized case (Figs. 12 and 13).

Figure 16 presents the behavior of hierarchical encoding with preprocessing when varying encryption key length. We show results for two different grid granularities and coverage values, with Gaussian zone distribution. As expected, the performance decreases when key length increases. However, the 1024-bit setting, which accord-ing to industry standards is sufficient for securing individuals' information, does not incur a steep increase in performance overhead. Gray encoding results exhibit similar behavior.

Figures 17 and 18 present the results when employing the parallel processing opti-mization. We used 2, 4 and 8 CPUs for computation. We considered both variable grid size for a fixed coverage of 6%, as well as variable coverage of alert zones for a fixed grid size of 256. The results show that a close-to-linear speedup can be obtained. For 8 CPUs for instance, the speedup is 7.2. This is a very encouraging outcome, and the query time is this way reduced to less than one second in the worst case. For median-scale settings of the grid size and coverage, we obtain absolute execution times of under 0.1 seconds per query. We emphasize that, although we only had available 8 CPUs for testing, the problem studied is embarrassingly parallel in nature, so the availabil-ity of a larger number of CPUs is likely to lead to close-to-linear speedup values as well.
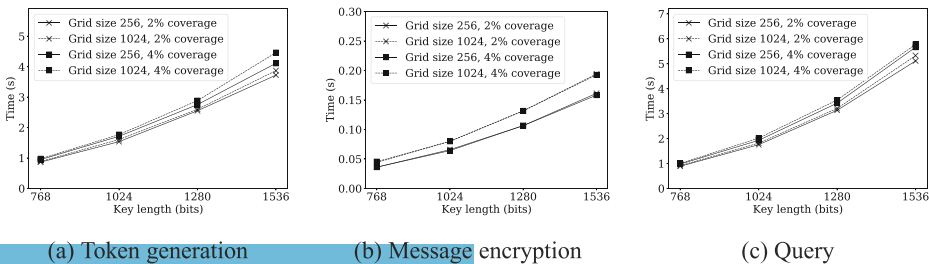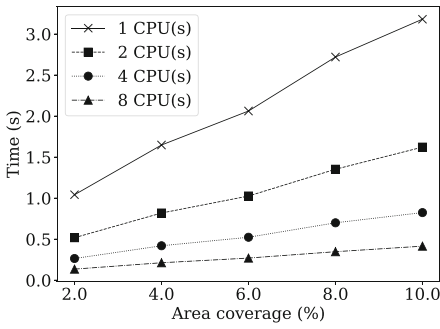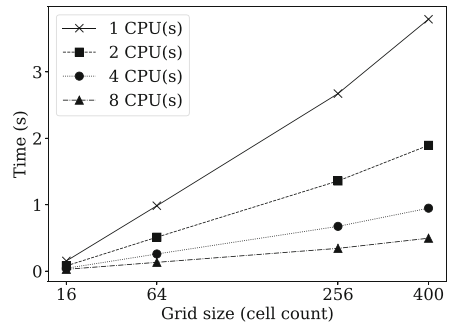

(a) Token generation   (b) Message encryption   (c) Query

Fig. 16 Hierarchical encoding results for Variable Key Length, Gaussian data
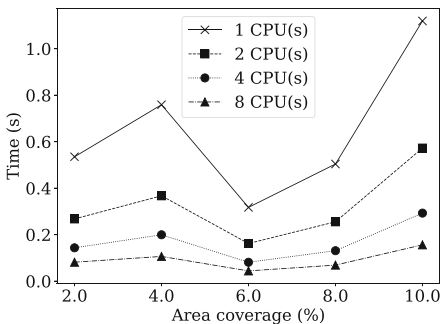
(a) Fixed area percentage   (b) Fixed grid size

Fig. 17 Parralel results on uniform data

6.4 Alert zone expansion evaluation

In this section, we evaluate the performance gain obtained by the alert zone expansion heuristic introduced in Section 5. We use the same settings as in the previous experiments, except that we allow a finer-grained partitioning of the space, to better evaluate the impact of the expansion heuristic. Specifically, we consider grids of granularity $d \times d$, where $d \in \{64, 128, 256\}$, resulting to a total number of grid cells of 4096, 16384 and 65536, respectively. We keep the same alert zone size ranging from 1% to 10% of data space size, but we consider three distinct shapes: square, rectangular (with a skew ratio of 2.5), and circular. The latter case is used to capture scenarios where there is an event epicenter, and individuals are notified if they are situated within a certain Euclidean distance of it. The resulting circular zone is mapped to the grid. This is representative for cases when mobile users are alerted to stay away from a dangerous location (e.g., a toxic gas spill). The alert zone expansion ratio α is varied within set $\{0.02, 0.04, 0.06, 0.08, \mathbf{0.10}\}$ (recall that a larger value results in a more significant privacy leakage, but is also likely to yield a higher performance gain).
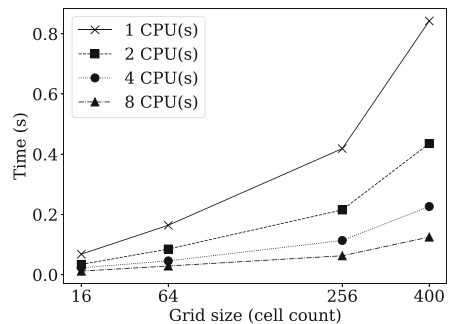
(a) Fixed area percentage   (b) Fixed grid size

Fig. 18 Parralel results on Gaussian data

Figure 19 shows the performance gain of expansion when varying the size of the initial alert zone. For ease of presentation, the gain is expressed as improvement factor ($\times$), i.e., the ratio between the matching time when there is no enlargement over the matching time when enlargement is used (a higher value represents a better performance gain). Each line in the graph corresponds to a different grid granularity d. First, we note that enlargement always results in improvement (the value is always greater than 1). Second, the improvement factor shows a general increasing trend with alert zone size (except for some random outcomes). This is explained by the fact that the enlargement factor α is expressed as a percent of initial alert zone size. When the initial alert zone is larger, the heuristic can select patches from more candidate cells. Finally, the improvement factor is larger for finer granularity cases (i.e., larger d). This is also due to the fact that the heuristic has more candidate patches to choose from. A finer granularity also allows the search boundary to advance slightly more. When cells are larger, including an extra cell may cause the α threshold to be exceeded, so the heuristic will not consider that cell for enlargement. We also note that the shape of the zone impacts significantly the gain. Specifically, a circular zone is better for expansion, since the heuristic does not favor any particular expansion direction. When the initial zone is circular (or to be precise, a circle aligned to the grid), the heuristic can bring into the zone cells from all directions, and therefore the amount of possible choices is increased. The zone is also likely to grow uniformly in all directions, leading to more compact tokens due to the binary representations of cells. Conversely, the rectangular case, which leads to the most skewed zones in terms of shape, performs the worst.

Figure 20 shows the improvement factor when varying the enlargement factor α. As expected, there is a clear increasing trend in execution time improvement. Since more cells are available as patch candidates, the heuristic is able to either completely eliminate some tokens, or significantly increase the number of wildcards in the remaining tokens through binary minimization. As in the previous experiment, we note that an increase in granularity d results in a higher improvement factor. Also, when the initial alert zone is circular, the highest improvement is obtained, with values of up to 9 times. The gain is less pronounced for rectangular alert zones, but the heuristic is still providing significant gains, with an improvement factor of up to 3 times.

In our final experiment, we measure the execution time of the zone enlargement heuristic. Figures 21 and 22 show the time required to compute the enlarged zone when varying initial alert zone size and enlargement factor α, respectively. An interesting trade-off emerges: as the granularity of the grid increases (i.e., finer grained grids), the improvement in token matching time increases (as seen in previous experiments), but at the same time the computation time for the enlarged zone grows. Furthermore, we emphasize that the token matching computational overhead can be parallelized, whereas zone enlargement
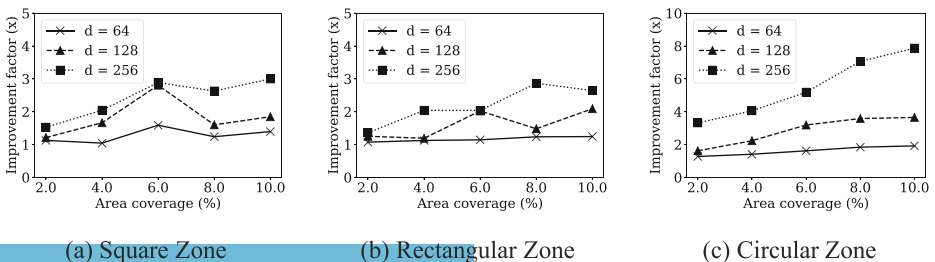

(a) Square Zone   (b) Rectangular Zone   (c) Circular Zone

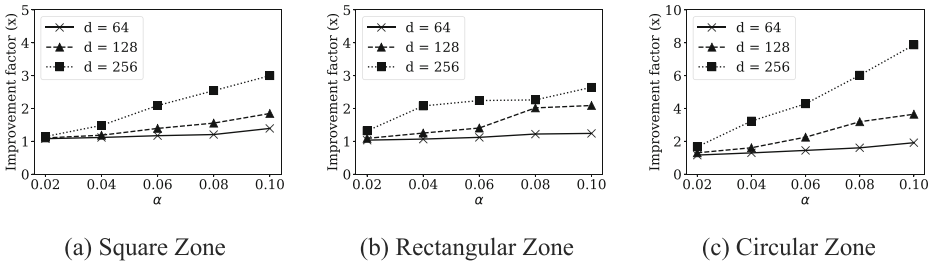**Fig. 19** Zone expansion improvement vs alert zone coverage

Fig. 20 Zone expansion improvement vs expansion factor α

computation is sequential in nature. The main reason why the zone enlargement computation is high for finer granularities is the quadratic increase in patch candidates, coupled with the relatively slow binary expression minimization step. Among different zone shapes, the circular shape takes the longest, due to the fact that it considers the most patch candidates within the given enlargement threshold α.

Nevertheless, we note that for coarser and moderate granularities ($d = 64$ and $d = 128$), the enlargement process is fast (less than half a second). Coupled with the significant improvement factors (ranging from 2 to 4 for granularities coarser than $d = 256$, as can be observed from Figs. 19 and 20), the heuristic can lead to very good overall execution time improvements. Furthermore, the enlargement cost is done once per zone, and remains the same regardless of the number of mobile users (i.e., ciphertexts to match against). Hence, as the user population grows, the performance gain of the heuristic (which is always a factor of the original zone evaluation time) will lead to linear gains in the number of users, whereas the enlargement computation overhead stays constant. We conclude that, overall, the zone enlargement heuristic is effective in reducing the matching overhead, even for small values of enlargement (i.e., only a small amount of privacy needs to be traded off for significant performance gains).

7 Related work

Location Privacy.

A significant amount of research focused on the problem of private location-based queries, where users send their coordinates to obtain nearby points of interest. Early work
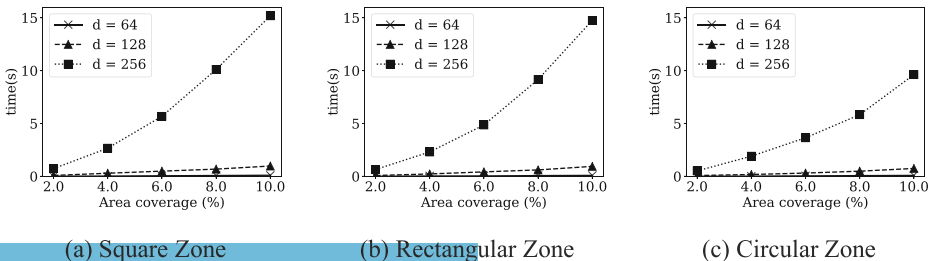

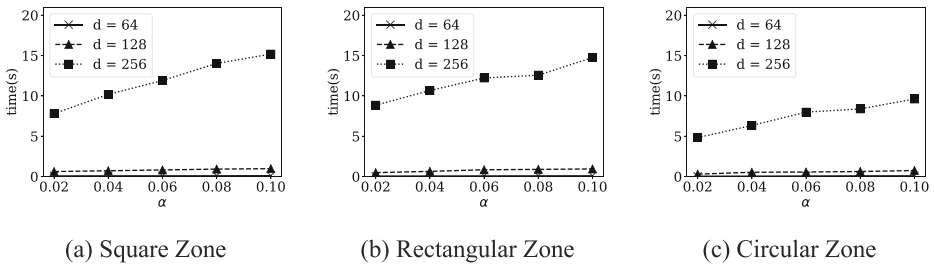Fig. 21 Zone expansion time vs alert zone coverage

Fig. 22 Zone expansion time vs expansion factor α

attempted to protect locations of real users by generating fake locations. For instance, in [27] the querying user sends to the server $k - 1$ fake locations to reduce the likelihood of identifying the actual user position.

However, fake locations can be detected using filtering techniques, which leaves the real users vulnerable.

A new direction of research started by [20] and continued by [14, 25, 34] relies on the concept of *Cloaking Regions (CRs)*. CR-based solutions implement the spatial k-anonymity (SKA) [25] paradigm. For each query, a trusted anonymizer service generates CRs that contain at least k *real* user locations.

If the resulting CRs are *reciprocal* [25], SKA guarantees privacy for snapshots of user locations. However, supporting continuous queries [8] requires generating large-sized CRs. In [10, 21], the objective is to prevent the association between users and sensitive locations. Users define privacy profiles [10] that specify their sensitivity with respect to certain *feature types* (e.g., hospitals, bars, etc.), and every CRs must cover a diverse set of sensitive and non-sensitive features.

In [26], the set of POI is first encoded according to a secret transformation by a trusted entity. A Hilbert-curve mapping (with secret parameters) transforms 2-D points to 1-D. Users (who know the transformation key) map their queries to 1D, and the processing is performed in the 1-D space. However, the mapping can decrease result accuracy, and the transformation may be vulnerable to reverse-engineering.

The problem with CR-based methods is that the underlying k-anonymity paradigm is vulnerable to background knowledge attacks. This is particularly a problem in the case of moving users, since trajectory information can be used to derive the identities behind reported locations. More recently, *differential privacy* [12], a provably secure model for semantic privacy, has been used for spatial data in [9]. However, differential privacy is only suitable for aggregate releases of data, and cannot handle processing of individual updates, as required by an alert system.

Closer to our work, a Private Information Retrieval (PIR) protocol is proposed in [16] for nearest-neighbor queries. The protocol is provably secure, and also uses cryptography. However, it considers a 'pull-based' approach, and assumes that the user already knows the location s/he wants to retrieve points of interest from. In contrast, our focus is on a 'push-based' notification service, where the PIR solution cannot be applied since the user is not aware of where the alert zones are.

Since the publication of [18], several works addressed processing on encrypted location data. In [37] and [22], two solutions are proposed for search on encrypted location data hosted at a cloud server. Both approaches rely on *symmetric searchable encryption* (SSE), where the client has access to the secret key of the transformation. The FastGeo system [38]

builds upon the concepts introduced in [37] and supports faster search under the same trust assumptions. However, the SSE setting is not appropriate in our problem setting, where large populations of mobile users subscribe to location-based alerts. If a single user colludes with the service provider, the security of the entire set of locations is compromised. This is a strong trust assumption, suitable for cases where there are relatively few clients, who can be throughly vetted. Our solution relies on asymmetric encryption, and mobile users only have access to the public key, which is used for encryption. No user is able to compromise the privacy of other participants.

Furthermore, all the above solutions build an index on encrypted data to speed up search performance. As shown in [39], the index structure can leak a lot of sensitive details about the data, even when fully encrypted (e.g., data distribution, or relative distance order among users). A similar approach that builds an R-tree on location data protected using homomorphic encryption is proposed in [28], with emphasis on IoT data, and on parallelizing computation in big data environments. The work in [29] is a position paper that looks at how some concepts similar to search on encrypted locations can be used for biomedical data, and also identifies other interesting type of queries that may be of interest, such as skyline queries.

A significant body of research focused on nearest-neighbor (NN) queries on encrypted data [13, 23, 24], culminating with the work in [39] which showed that the most secure and efficient way to answer NN queries on encrypted data is through materialization of results and encryption of the resulting structure. All these works consider the symmetric encryption setting, hence they rely on trust assumptions that are too strong for our proposed location alert system.

Searchable Encryption. One of the earliest works that coined the concept of searchable encryption was [36], which proposed provably secure cryptographic techniques for keyword search. Only exact matches of keywords were supported. Later in [5], the set of search predicates supported was extended to comparison queries. However, the resulting solution could not be easily extended to conjunctions of conditions, without a considerable increase in ciphertext and token size. The work in [6] further extended the set of supported predicates to subset queries, as well as conjunctions of equality, comparison and subset queries with small ciphertext and token size. The authors of [6] also introduced HVE, which we employ as a building block in our solutions for private location-based alert systems. HVE protects the privacy of the encrypted messages received from users, but assumes that the token information (e.g., alert zones) is public. The more recent work in [2] extends HVE to also protect the tokens. However, the solution is more expensive.

8 Conclusion

We proposed a system for secure location-based alerts which utilizes searchable encryption. We introduced two alternate data encodings that allow efficient application of cryptographic primitives for search on encrypted data (namely HVE). Furthermore, we devised performance optimizations that reduce the overhead of searchable encryption, which is notoriously expensive. We also devised a heuristic that enlarges alert zones by a small factor in order to reduce matching time, thus achieving a tunable performance-privacy trade-off. The experimental evaluation results show that searchable encryption can be made practical with careful system design and optimizations.

In future work, we plan to investigate more advanced data and query encoding techniques (beyond regular grids) that will allow us to securely alert users with even lower overhead. We also plan to study other types of matching semantics beyond range queries (e.g., nearest-neighbors, top-k).

Appendix

A Primer on HVE Encryption

HVE is built on top of a symmetrical bilinear map of composite order [4], which is a function $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ 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

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

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.

References

1. Bitner JR, Ehrlich G, Reingold EM (1976) Efficient generation of the binary reflected gray code and its applications. Commun ACM 19(9):517–521
2. Blundo C, Iovino V, Persiano G (2009) Private-key hidden vector encryption with key confidentiality. In: Proceedings of the 8th international conference on cryptology and network security, pp 259–277
3. Boneh D, Crescenzo GD, Ostrovsky R, Persiano G (2003) Public key encryption with keyword search. In: EUROCRYPT 2004, volume 3027 of LNCS
4. Boneh D, Goh E-J, Nissim K (2005) Evaluating 2-dnf formulas on ciphertexts. In: Proceedings of the 2nd international conference on theory of cryptography, pp 325–341
5. Boneh D, Sahai A, Waters B (2006) Fully collusion resistant traitor tracing with short ciphertexts and private keys. In: EUROCRYPT 2006, volume 4004 of LNCS, pp 573–592
6. Boneh D, Waters B (2007) Conjunctive, subset, and range queries on encrypted data, pp 535–554
7. Brinkhoff T (2002) A framework for generating network-based moving objects. GeoInformatica 6(2):153–180
8. Chow C-Y, Mokbel MF (2007) Enabling Private Continuous Queries for Revealed User Locations. In: SSTD, pp 258–275
9. Cormode G, Procopiuc C, Shen E, Srivastava D, Yu T (2012) Differentially private spatial decompositions. In: ICDE, pp 20–31
10. Damiani M, Bertino E, Silvestri C (2008) PROBE: an Obfuscation System for the Protection of Sensitive Location Information in LBS. Technical Report 2001-145 CERIAS
11. Dwork C (2010) Differential privacy in new settings. In: SODA, pp 174–183
12. Dwork C, McSherry F, Nissim K, Smith A (2006) Calibrating noise to sensitivity in private data analysis. In: TCC, pp 265–284
13. Elmehdwi Y, Samanthula BK, Jiang W (2013) Secure k-nearest neighbor query over encrypted data in outsourced environments. In: IEEE international conference on data engineering (ICDE), pp 664–675
14. Gedik B, Liu L (2005) Location privacy in mobile systems: A personalized anonymization model. In: Proc. of ICDCS, pp 620–629
15. Gedik B, Liu L (2008) Protecting location privacy with personalized k-Anonymity: Architecture and algorithms. IEEE TMC 7(1):1–18
16. Ghinita G, Kalnis P, Khoshgozaran A, Shahabi C, Tan KL (2008) Private Queries in Location Based Services: Anonymizers are not Necessary. In: Proceedings of international conference on management of data (ACM SIGMOD)
17. Ghinita G, Kalnis P, Skiadopoulos S (2007) PRIVE: Anonymous location-based queries in distributed mobile systems. In: WWW
18. Ghinita G, Rughinis R (2014) An efficient privacy-preserving system for monitoring mobile users: Making searchable encryption practical. In In Proc. of Intl, Conference on Data and Application Security and Privacy (CODASPY), pp 321–332
19. Goldreich O (2004) The foundations of cryptography, vol 2. Cambridge University Press, Cambridge
20. Gruteser M, Grunwald D (2003) Anonymous usage of location-based services through spatial and temporal cloaking. In: USENIX MobiSys
21. Gruteser M, Liu X (2004) Protecting privacy in continuous location-tracking applications. IEEE Secur Priv 2:28–34
22. Guo R, Qin B, Wu Y, Liu R, Chen H, Li C (2019) Mixgeo: Efficient secure range queries on encrypted dense spatial data in the cloud. In: Proceedings of the international symposium on quality of service, IWQoS '19
23. Hashem T, Kulik L, Zhang R (2010) Privacy preserving group nearest neighbor queries. In: IEEE international conference on data engineering (ICDE), vol 01, pp 489–500
24. Hu H, Xu J, Ren C, Choi B (2011) Processing private queries over untrusted data cloud through privacy homomorphism. In: IEEE international conference on data engineering (ICDE), pp 601–612

25. Kalnis P, Ghinita G, Mouratidis K, Papadias D (2007) Preserving location-based identity inference in anonymous spatial queries. IEEE TKDE, 19(12)
26. Khoshgozaran A, Shahabi C (2007) Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In: SSTD
27. Kido H, Yanagisawa Y, Satoh T (2005) An anonymous communication technique using dummies for location-based services. In: International conference on pervasive services (ICPS), pp 88–97
28. Limkar S, Jha R (2019) Computing over encrypted spatial data generated by iot. Telecommun Sys 70:193–229
29. Liu J, Xiong L (2015) Secure similarity queries: Enabling precision medicine with privacy. In: Biomedical data management and graph online querying
30. Liu Q, Wang G, Wu J (2012) Secure and privacy preserving keyword searching for cloud storage services. J Netw Comput Appl 35(3):927–933
31. Lynn B (2007) On the implementation of pairing-based cryptography. Stanford University, PhD thesis
32. McGeer P, Sanghavi J, Brayton R, Vincentelli AS (1993) Espresso-signature: A new exact minimizer for logic functions. In: Proceedings of the 30th international design automation conference, pp 618–624
33. Miller VS (2004) The weil pairing, and its efficient calculation. J Cryptol 17(4):235–261
34. Mokbel MF, Chow C-Y, Aref WG (2006) The new casper: Query processing for location services without compromising privacy. In: VLDB
35. Samet H (1984) The quadtree and related hierarchical data structures. ACM Comput Surv 16(2):187–260
36. Song DX, Wagner D, Perrig A (2000) Practical techniques for searches on encrypted data. In: IEEE symposium on security and privacy
37. Wang B, Li M, Wang H (2016) Geometric range search on encrypted spatial data. IEEE Trans Inf Foren Secur 11(4):704–719
38. Wang B, Li M, Xiong L (2019) Fastgeo: Efficient geometric range queries on encrypted spatial data. IEEE Trans Depend Secure Comput 16(2):245–258
39. Yao B, Li F, Xiao X (2013) Secure nearest neighbor revisited. In: Proc. of intl, conf. on data engineering, pp 733–744

**Gabriel Ghinita** is an Associate Professor with the Department of Computer Science, University of Massachusetts, Boston. His research interests lie in the area of data security and privacy, with focus on privacy-preserving transformation of microdata, private queries in location based services and privacy-preserving sharing of sensitive datasets. Prior to joining University of Massachusetts, Dr. Ghinita was a research associate with the Cyber Center at Purdue University, and a member of the Center for Education and Research in Information Assurance and Security (CERIAS). Dr. Ghinita served as reviewer for top journals and conferences such as IEEE TPDS, IEEE TKDE, IEEE TMC, VLDBJ, VLDB, WWW, ICDE and ACM SIGSPATIAL GIS.

Kien Nguyen is a graduate student pursuing his Ph.D. degree in Computer Science in the Viterbi School of Engineering at the University of Southern California, under the supervision of Professor Cyrus Shahabi. He received his Bachelor of Science from Hanoi University of Science and Technology (Vietnam) in 2013. Upon graduation, he was a Research Scientist in the Research and Development Department of VNG Corporation in Vietnam, conducting research on data mining and recommender systems for online social networks. He is interested in spatial data, privacy, machine learning, data mining, and their applications.
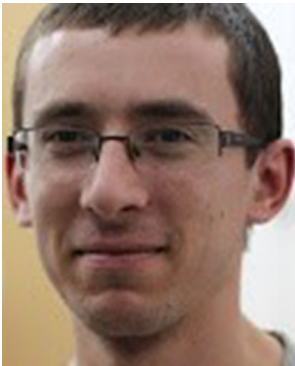
Mihai Maruseac recently obtained a PhD in Computer Science from the University of Massachusetts Boston. He holds BS and MS degrees in Computers Science from "Politehnica" University of Bucharest. His research interests span location privacy and privacy-preserving data mining. His work focuses on differentially private algorithms for mining trajectory data, and efficient searchable encryption techniques for location-based queries.

🖄 Springer

Cyrus Shahabi is a Professor of Computer Science and Electrical Engineering and the Director of the NSF's Integrated Media Systems Center (IMSC) at the University of Southern California (USC). He is also the director of the Informatics Program at USC's Viterbi School of Engineering. He received his B.S. in Computer Engineering from Sharif University in 1989 and then his M.S. and Ph.D. Degrees in Computer Science from USC in May 1993 and August 1996, respectively. He authored two books and more than two hundred research papers in the areas of databases, GIS and multimedia.

Dr. Shahabi is a fellow of IEEE, and a recipient of the ACM Distinguished Scientist award in 2009, the 2003 U.S. Presidential Early Career Awards for Scientists and Engineers (PECASE), and the NSF CAREER award in 2002.