An Efficient Privacy-Preserving System for Monitoring Mobile Users: Making Searchable Encryption Practical

Gabriel Ghinita
University of Massachusetts, Boston
gabriel.ghinita@umb.edu

Razvan Rughinis
Politehnica University, Bucharest
razvan.rughinis@cs.pub.ro

ABSTRACT

Monitoring location updates from mobile users has important applications in several areas, ranging from public safety and national security to social networks and advertising. However, sensitive information can be derived from movement patterns, so 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 when an event of interest occurs nearby. Currently, such functionality is 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)*, and 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 also implement two optimizations that further improve performance: materialization of results to expensive mathematical operations, and parallelization. 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.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms

Security, Experimentation

Keywords

Location Privacy, Pairing-based Cryptography

1. INTRODUCTION

Modern mobile devices with positioning capabilities (e.g., GPS) allow users to be informed about events that occur in their proximity. Several classes of applications benefit from the large-scale availability of location data, ranging from public safety 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 condition, 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 retail store sales event is underway nearby.

The typical architecture of such a system uses a server that collects updates from the users and checks whether the alert condition is met. Such a service is provided by a commercial entity that is not fully trusted. Collection of user trajectories at a commercial site introduces serious privacy concerns, as sensitive personal information may be derived from a person's whereabouts [19, 16]. 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. Various syntactic privacy models have been proposed [19, 15, 28, 17] that perform some sort of generalization of location before sharing, but these have been proven to be vulnerable, especially in the presence of background knowledge [16]. Furthermore, semantic privacy models such as differential privacy [13, 12, 10] 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 [6, 24, 2]. These are broadly referred to as *searchable encryption* functions, since they allow the evaluation of certain types of queries without requiring decryption. Such encryption systems are asymmetric, i.e., they employ a secret key *SK* and a public key *PK* pair.

Figure 1 illustrates the envisioned system for location-based alerts using searchable encryption. The architecture has three types of entities: *(i)* the users, who send encrypted location updates using *PK*; *(ii)* a *trusted authority (TA)* which generates *tokens* for spatial search predicates using secret key SK; and *(iii) the server (S)* that collects updates from users and evaluates the predicates on 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.

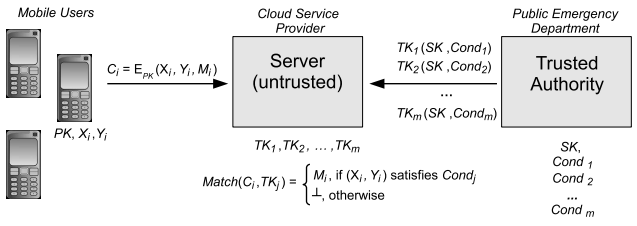


Figure 1: Location-based Alert System

CODASPY'14, March 3–5, 2014, San Antonio, Texas, USA.

http://dx.doi.org/10.1145/2557547.2557559.

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

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 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, it uses its current coordinates (X_i, Y_i) as index, and performs encryption as $E_{PK}((X_i, Y_i), M_i)$. At the server, 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]. The method allows queries on ciphertexts such as exact match, range and subset queries. HVE uses bilinear maps on groups of composite order [18] as mathematical foundation and makes extensive use of expensive operations such as bilinear maps pairings and exponentiations with large integers. As a result, HVE is very expensive and scales poorly. Later in Section 5, 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 searchable encryption to the domain of private search with spatial predicates. Our specific contributions are:

i. We devise specific constructions that allow application of HVE to the problem of location-based alert systems with a reduced number of 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 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.

The rest of the paper is organized as follows: 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. Section 5 contains the experimental evaluation results, followed by a survey of related research in Section 6. Finally, Section 7 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 a number of n users $\{u_1, \ldots, u_n\}$ move within a two-dimensional square unit space $[0, 1] \times [0, 1]$. Users continuously report their

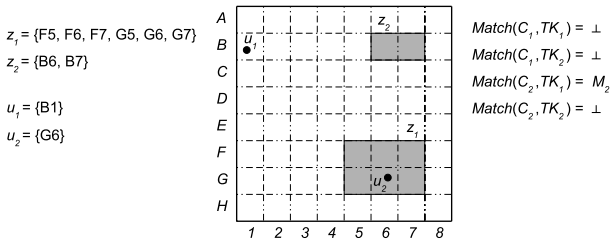


Figure 2: System Model ($n = 2$, $m = 2$, $d = 8$)

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. 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 architecture follows the model represented in Figure 1, and consists of three types of entities:

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

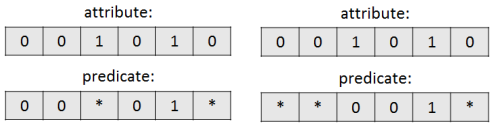
ii. The **Trusted Authority (TA)** is a trusted entity that decides which are the alert zones, and creates for each zone a search *token* that allows to check privately if a user location falls within the alert zone or not[1].

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

The *privacy requirement* of the system dictates that the server must not learn any information about the user locations, other than what can be derived from the match outcome, i.e., whether the user is in a particular alert zone or not. In case of a successful match, the server S learns that user u is enclosed by zone z. In case of a non-match, the server S learns only that the user is outside the zone z, but no additional location information is disclosed. 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. In an alternate setting, the TA may be a national security agency which investigates potentially terrorist activities near a given location that it has information on. Cooperating civilians that are in the area may want to volunteer some

[1]Later in Section 3 we show how a single token can serve multiple zones, which increases performance.



(a) Match (b) Non-Match

Figure 3: Predicate evaluation on ciphertexts with HVE

information, such as Wi-Fi packet traces captured by their devices. However, they do not want to disclose their location when they are outside the suspicious zone.

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 monitors privately 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.

Finally, we emphasize that the alert zones are *not* private. This is a reasonable assumption, since in practice the existence of emergency zones is public information that is broadly available. If it is important to protect the location of the zones, then a related flavor of HVE which protects token information can be used [2]. However, dealing with this scenario is outside the scope of this work.

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 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.

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

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

We provide additional mathematical background on HVE encryption and its operations in Appendix A.

3. PROPOSED SPATIAL HVE APPROACHES

In this section, we show how to use HVE for location-based alert systems. 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 5. To bring down the overhead of HVE, we propose in Section 3.2 a *hierarchical encoding* technique, which reduces the amount of cryptographic primitives required during search. Next, in Section 3.3 we further refine hierarchical encoding and devise the *Gray encoding*, which achieves superior computation savings.

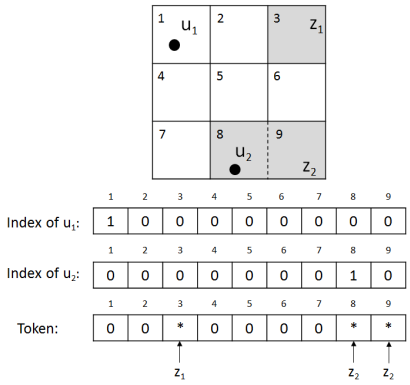


Figure 4: Naive Baseline Encoding

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 searchable encryption for location-based alerts is to use an HVE with width $l = d^2$. The encoding of the HVE is performed as follows:

- When user u enclosed by grid cell i reports its location, it uses in the encryption process 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. This is equivalent to a subset query, and signifies that the user location is enclosed in some alert zone.

Consider the example in Figure 4, where $d = 3$. We have a number of 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 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-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.

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

Algorithm 2: Baseline Encoding

p = ID of grid cell enclosing user u
A = set of grid cell IDs that make up the alert zone
User u: Set $I = [I_1 : I_{d^2}]$, $I_p = 1$ and $\forall i \neq p, I_i = 0$
User u: Send encrypted I to Server S
TA : Set $T = [T_1 : T_{d^2}]$, $T_i = *$ 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 $NotMatch$

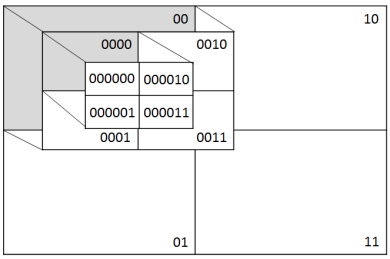


Figure 5: Hierarchical partitioning on three levels

number of cells is not appropriate. 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 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, in doing so, 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. To address this issue, 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] [29]. 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 $j + 1$ (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 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). Going one step further with minimization, 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 [26].

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

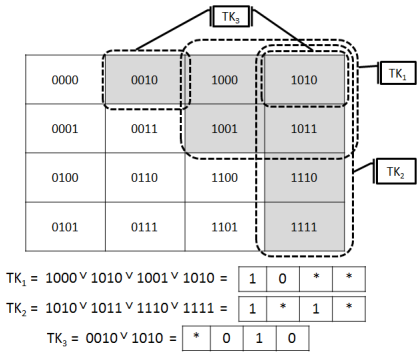


Figure 6: Hierarchical encoding and token aggregation

Consider for instance the example from Figure 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 (encrypted) user location, 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 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 representations of cell identifiers within the zone. Next, the TA computes the minimized binary expression equivalent to the logical 'OR' of all identifiers 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 5. Algorithm 3 provides the hierarchical encoding pseudocode.

Algorithm 3: Hierarchical Encoding

p = ID of grid cell enclosing user u
A = set of grid cell IDs that make up the alert zone
User u: Set $I = [I_1 : I_{2\log_2 d}]$, 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 $TK = [TK_1 : TK_{2\log_2 d}]$, 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:

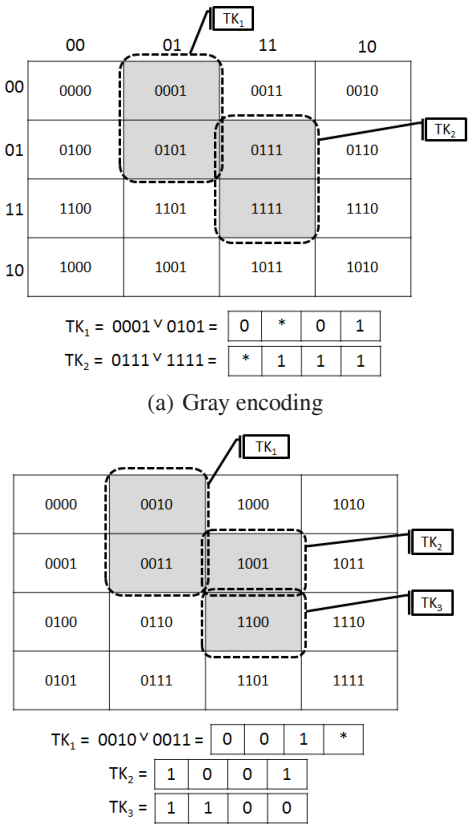
$$G_1 = (0, 1)$$

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

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

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



(a) Gray encoding

(b) Hierarchical encoding

Figure 7: Token Aggregation: Gray vs hierarchical encoding

consists of four cells. The two digits leading each row in the Gray encoding diagram (Figure 7(a)) 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 7(b) 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 experimentally in Section 5, Gray encoding achieves more effective aggregation, especially in the case of skewed data (e.g., Gaussian distribution of alert zones).

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 in Section 5, 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 in-

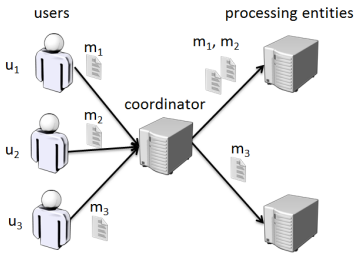


Figure 8: Parallel computation model

cur 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 of them 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 [27]. Typically, for each such operation, it is required to compute several line equations. In [25] 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 receives a large number of messages that must be matched against all tokens. 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. A message-passing system can scale much easier to a large number of CPUs, without the need for expensive hardware (as is the case for shared-memory machines). As programming environment, we chose the *Message Passing Interface (MPI)* which is a freely-available software. One master MPI 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.

Figure 8 illustrates the envisioned computing infrastructure. Users u_1, u_2 and u_3 send messages m_1, m_2 and m_3 to the coordinator. m_1 and m_2 are allocated to the first processing unit and m_3 to the second. As a result, the messages are processed in parallel, and the time between message generation and matching is reduced.

The infrastructure described above has a potential limitation in the case when a very small number of users subscribe to the system. If processing a message from a user is always finished before receiving messages from another, then only one single processing entity will be active at a time, whereas the others would idle. One solution could be to assign multiple CPUs to process the same message, but on different tokens. In order to do so, the central coordinator must broadcast the user message to each machine. Next, each processing unit will only perform matching with respect to a disjoint partition of the tokens. However, if one processing entity evaluates one token successfully, it will have to signal this event to other slaves, because there is no need to evaluate all other tokens for that particular zone. As a result, communication between the slaves becomes necessary, creating additional overhead and decreasing speedup. Nevertheless, an alert system is likely to have a large number of users, hence the architecture described in Figure 8 where each message is handled by a single CPU is sufficient.

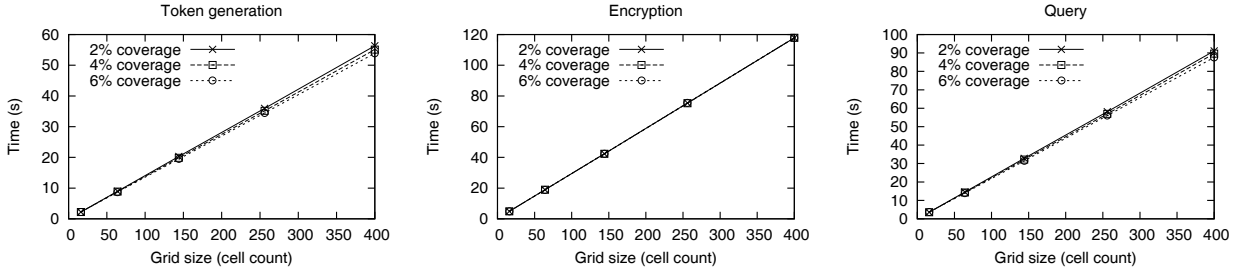
5. EXPERIMENTAL EVALUATION

We implemented a C++ 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*[3] [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 2% to 6%, and we denote this parameter as *coverage*. We consider a regular grid partitioning the two-dimensional space with size ranging from 16 to 400. The HVE cryptographic functions were implemented using the Gnu MP library[4] and the Pairing-Based Cryptography library[5] from Stanford University. We use key lengths of 768, 1024 (default value), 1280 and 1536 bits. Throughout the experimental evaluation, we focus on the operations of encryption, token generation and query, and we do not consider the setup (key generation) phase. We have found that this phase always requires less than 2 seconds, and it is a one-time process, hence does not impact system performance. The experimental testbed consisted of a cluster with

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

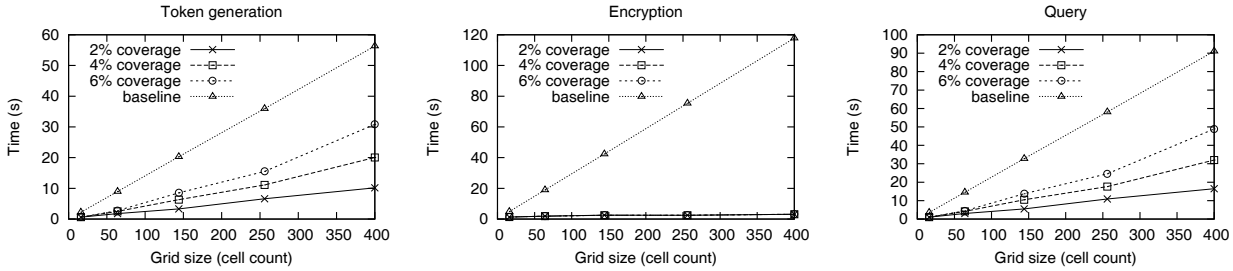
[4]Available online at http://gmplib.org/

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



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

Figure 9: Baseline encoding results



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

Figure 10: Hierarchical encoding results on uniform data

8 machines, each having an Intel Xeon 2.53 GHz CPU and 32GB of RAM, running Red Hat Enterprise Linux and using a network connection of 10Gbps. For the parallel tests, we used as programming environment Open MPI 1.5.1.

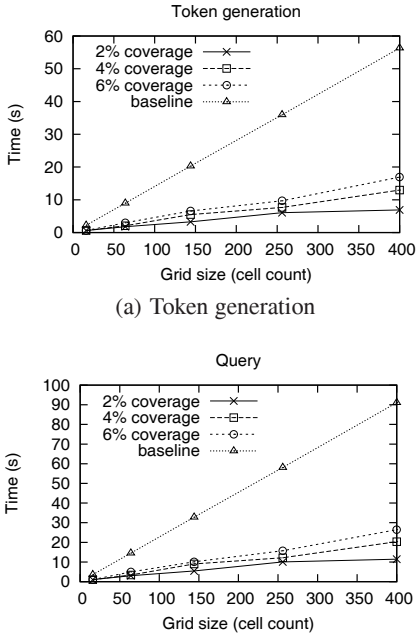
5.1 Baseline Evaluation

We evaluate the performance of the baseline method introduced in Section 3.1, thus justifying our claim that direct application of searchable encryption to location-based alert systems is not practical. Figure 9 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.

The token generation ranges from 10 to 60 seconds. Even though this is 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. Even for moderate grid sizes, it takes 40 to 80 seconds to generate a single encrypted update! Furthermore, the time required at the server to process a single user update (i.e., perform matching against all alert zones) ranges between 30 and 60 seconds for moderate grid sizes, reaching as high as 100 seconds for the largest grid size.

5.2 Hierarchical and Gray Encoding

Next, we evaluate the hierarchical encoding performance against the baseline. Figures 10 and 11 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 3 seconds, in contrast with 120 seconds for the baseline (Figure 9(b)). Recall that alert zones do not influence encryption, so a single line is present in Figure 10(b). Encryption is also independent of alert zone distribution, so we do not show encryption in Figure 11.

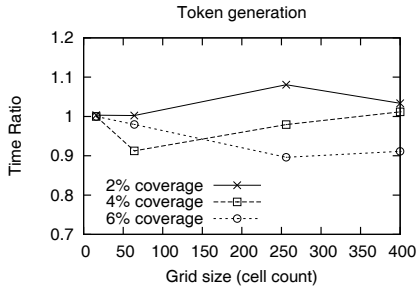


(a) Token generation

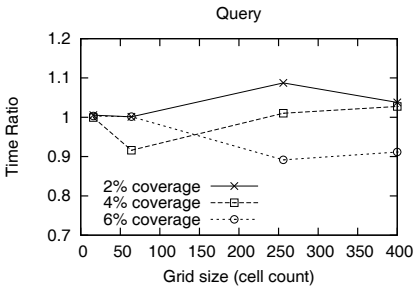
(b) Query

Figure 11: Hierarchical encoding results on Gaussian data

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.

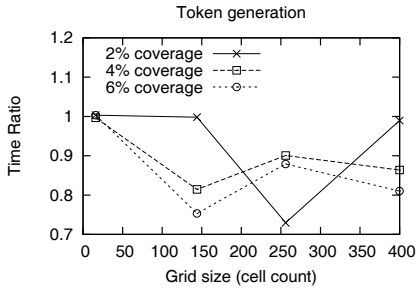


(a) Token generation

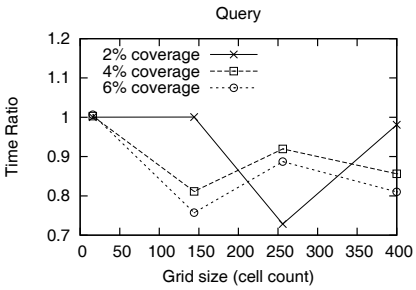


(b) Query

Figure 12: Gray vs. hierarchical encoding on uniform data



(a) Token generation



(b) Query

Figure 13: Gray vs. hierarchical encoding on Gaussian data

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 1.5 times). Although the absolute execution times are still high, hierarchical encoding significantly outperforms the baseline. Later in Section 5.3 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. Figures 12 and 13 present a comparison between the performance of the two encodings (since the encryption step is the same for hierarchical and Gray encodings, we omit that graph; the results are the same as in Figure 10(b)). 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.

For uniform data, 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 where alert zones are clustered, Gray encoding favors aggregation of cells. In practice, as alert zones are likely to be clustered, Gray can bring significant performance benefits, of up to 30%. The gain of Gray is higher as the coverage of alert zones increases, and there are more alternatives for cell aggregation.

5.3 Optimization Effect

In this section, we evaluate the performance of the proposed techniques when incorporating the performance optimizations discussed in Section 4. We show that, with the optimizations in place, the absolute cost of the proposed system for private location-based alerts can be brought down to a reasonable level, which can be practical in a commercial computing environment such as the cloud.

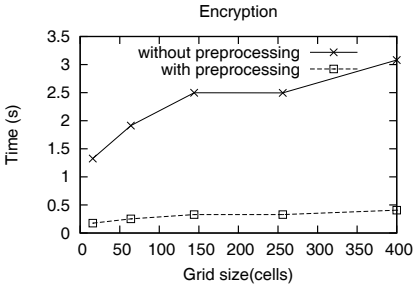
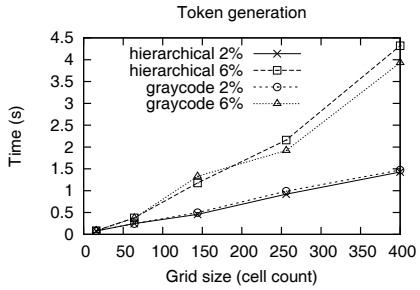


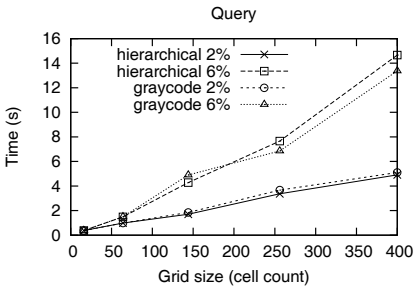
Figure 14: Encryption Time with preprocessing

First, we show the effect of incorporating preprocessing to precompute and re-use some of the results to expensive mathematical operations, such as exponentiations with large numbers. Figure 14 presents the gain in encryption time (recall that encryption time is the same for hierarchical and Gray encoding, and is independent of alert zone distribution). The time is reduced to levels of at most half a second. For coarser-grained grids, this time is as low as 0.2 seconds. Recall that, having a low encryption time is very important, since this operation is performed at the mobile user device.

Figures 15 and 16 present the absolute token generation and query times for both proposed encoding techniques, and we choose the extreme values of the alert zone coverage scale, namely 2% and 6%. Token generation computation requirements range from half a second to 4 seconds. For Gaussian distribution of alert zones, which is more relevant for real-life scenarios, the time is lower, as aggregation is more effective. As expected, Gray slightly outperforms hierarchical encoding, especially for the Gaussian data. Also, as the coverage increases, more tokens are required to represent a zone, so the generation time increases. We believe that such times are reasonable in practice, especially since creation of alert zones is not a frequent event in the system operation. In terms of

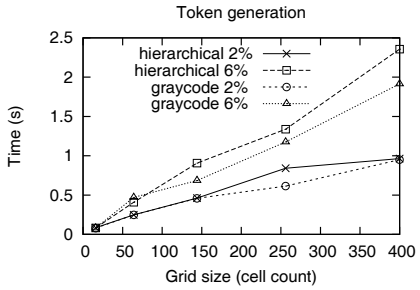


(a) Token generation

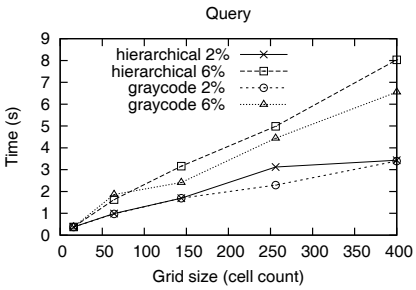


(b) Query

Figure 15: Effect of preprocessing on uniform data



(a) Token generation



(b) Query

Figure 16: Effect of preprocessing on Gaussian data

querying (i.e., matching) at the server, the execution times are approximately cut in half compared to the non-optimized case (Figures 10 and 11). We remark a similar increasing trend in execution time as the grid size and coverage increase. The Gray encoding maintains its advantage, especially for Gaussian distribution of alert zones.

In Figure 17 we present the behavior of hierarchical encoding with preprocessing when the length of the encryption key is varied. 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 according to industry standards is sufficient for securing individuals' information, does not incur a steep increase in performance overhead. Gray encoding results exhibit similar behavior.

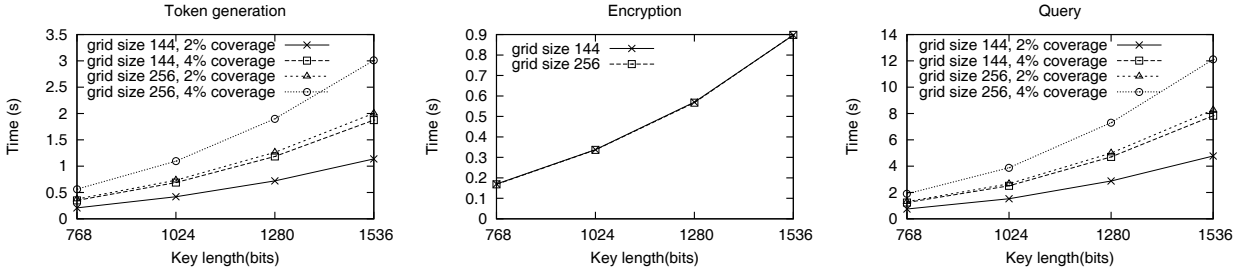
Despite the gains of incorporating preprocessing, the absolute times for queries at the server are quite high. Given that an alert system is likely to serve a large number of users, scalability is a key concern. Reducing further the matching time is essential, so we use parallel processing to solve this problem. Figures 18 and 19 present the results when employing the parallel processing optimization. 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 250. The results show that a close-to-linear speedup can be obtained. For 8 CPUs for instance, the speedup is 7.3. 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 0.15 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 availability of a larger number of CPUs is likely to lead to close-to-linear speedup values as well.

6. RELATED WORK

Location Privacy. The problem of location privacy has been extensively studied over the past decade, as it became clear that mobile users' movement patterns can disclose sensitive information about individuals. A significant amount of research focused on the problem of private location-based queries, where users send their coordinates to obtain points of interest in their proximity. Some of the earlier work attempted to protect locations of real users by generating fake locations. For instance, in [23] the querying user sends to the server $k-1$ fake locations to reduce the likelihood of identifying the actual user position. *SpaceTwist* [31] performs a multiple-round incremental range query protocol, based on a fake *anchor* location that hides the user coordinates. In [8], a random cloaking region that encloses the user is generated. However, the fake locations can be detected using various filtering techniques, which leaves the real users vulnerable.

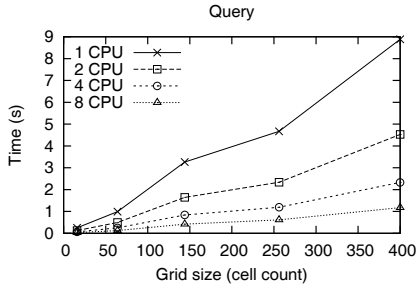
A new direction of research started by [19] and continued by [14, 21, 28] relies on the concept of *Cloaking Regions (CRs)*. CR-based solutions implement the spatial k-anonymity (SKA) [21] paradigm. For each query, a trusted anonymizer service generates CRs that contain at least k *real* user locations. If the resulting CRs are *reciprocal* [21], SKA guarantees privacy for snapshots of user locations. However, supporting continuous queries [9] requires generating large-sized CRs. In [20, 11], the objective is to prevent the association between users and sensitive locations. Users define privacy profiles [11] 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 [22], 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* [13], a

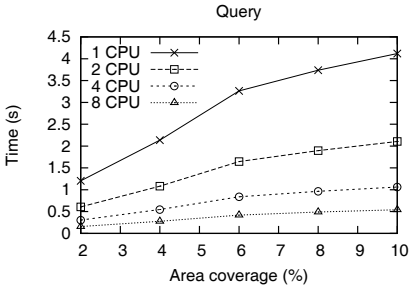


(a) Token generation (b) Message encryption (c) Query

Figure 17: Hierarchical encoding results for Variable Key Length, Gaussian data

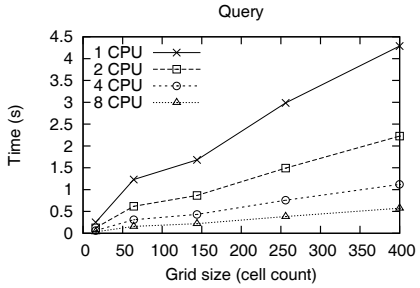


(a) Fixed area percentage

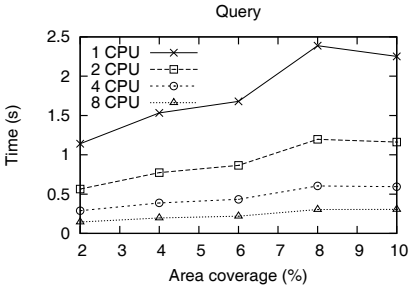


(b) Fixed grid size

Figure 18: Parralel results on uniform data



(a) Fixed area percentage



(b) Fixed grid size

Figure 19: Parralel results on Gaussian data

provably secure model for semantic privacy, has been used for spatial data in [10]. 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.

Searchable Encryption. One of the earliest works that coined the concept of searchable encryption was [30], 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.

7. CONCLUSION

In this paper, we proposed a provably-secure system for location-based alerts which uses searchable encryption. We introduced two alternate data encodings that allow the efficient application of cryptographic primitives for search on encrypted data (namely HVE), and we also devised performance optimizations that reduce the overhead of searchable encryption, which is notoriously expensive. To the best of our knowledge, this is the first attempt to use searchable encryption in the context of processing location data, and the experimental evaluation results show that searchable encryption can be made practical with careful system design and optimizations.

In future work, we plan to extend our solutions to more complex types of notification conditions. Currently, only range query semantics are supported, where a user is notified if its location falls within an alert zone. We will investigate techniques that also allow

distance-based search predicates. In addition, we will research how to check conditions that depend on the locations of multiple users, and how to protect efficiently the privacy of alert zones as well.

Acknowledgments. This work has been supported by NSF award CNS-1111512.

8. REFERENCES

[1] J. R. Bitner, G. Ehrlich, and E. M. Reingold. Efficient generation of the binary reflected gray code and its applications. *Commun. ACM*, 19(9):517–521, Sept. 1976.

[2] C. Blundo, V. Iovino, and G. Persiano. Private-key hidden vector encryption with key confidentiality. In *Proceedings of the 8th International Conference on Cryptology and Network Security*, pages 259–277, 2009.

[3] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT 2004, volume 3027 of LNCS*, 2003.

[4] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proceedings of the Second international conference on Theory of Cryptography*, pages 325–341, 2005.

[5] D. Boneh, A. Sahai, and B. Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT 2006, volume 4004 of LNCS*, pages 573–592, 2006.

[6] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Proceedings of the 4th conference on Theory of cryptography*, pages 535–554, 2007.

[7] T. Brinkhoff. A Framework for Generating Network-based Moving Objects. *GeoInformatica*, 6(2):153–180, 2002.

[8] R. Cheng, Y. Zhang, E. Bertino, and S. Prahbakar. Preserving User Location Privacy in Mobile Data Management Infrastructures. In *Privacy Enhancing Technologies (PET)*, 2006.

[9] C.-Y. Chow and M. F. Mokbel. Enabling Private Continuous Queries for Revealed User Locations. In *SSTD*, pages 258–275, 2007.

[10] G. Cormode, C. Procopiuc, E. Shen, D. Srivastava, and T. Yu. Differentially private spatial decompositions. In *ICDE*, pages 20–31, 2012.

[11] M. Damiani, E. Bertino, and C. Silvestri. PROBE: an Obfuscation System for the Protection of Sensitive Location Information in LBS. Technical Report 2001-145, CERIAS, 2008.

[12] C. Dwork. Differential privacy in new settings. In *SODA*, pages 174–183, 2010.

[13] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[14] B. Gedik and L. Liu. Location Privacy in Mobile Systems: A Personalized Anonymization Model. In *Proc. of ICDCS*, pages 620–629, 2005.

[15] B. Gedik and L. Liu. Protecting Location Privacy with Personalized k-Anonymity: Architecture and Algorithms. *IEEE TMC*, 7(1):1–18, 2008.

[16] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K. L. Tan. Private Queries in Location Based Services: Anonymizers are not Necessary. In *Proceedings of International Conference on Management of Data (ACM SIGMOD)*, 2008.

[17] G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVE: Anonymous Location-based Queries in Distributed Mobile Systems. In *WWW*, 2007.

[18] O. Goldreich. The Foundations of Cryptography, Volume 2. *Cambridge University Press*, 2004.

[19] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *USENIX MobiSys*, 2003.

[20] M. Gruteser and X. Liu. Protecting Privacy in Continuous Location-Tracking Applications. *IEEE Security and Privacy*, 2:28–34, 2004.

[21] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preserving Location-based Identity Inference in Anonymous Spatial Queries. *IEEE TKDE*, 19(12), 2007.

[22] A. Khoshgozaran and C. Shahabi. Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In *SSTD*, 2007.

[23] H. Kido, Y. Yanagisawa, and T. Satoh. An anonymous communication technique using dummies for location-based services. In *International Conference on Pervasive Services (ICPS)*, pages 88–97, 2005.

[24] Q. Liu, G. Wang, and J. Wu. Secure and privacy preserving keyword searching for cloud storage services. *J. Network Computing Applications*, 35(3):927–933, May 2012.

[25] B. Lynn. *On the Implementation of Pairing-Based Cryptography*. PhD thesis, Stanford University, 2007.

[26] P. McGeer, J. Sanghavi, R. Brayton, and A. S. Vincentelli. Espresso-signature: a new exact minimizer for logic functions. In *Proceedings of the 30th international Design Automation Conference*, pages 618–624, 1993.

[27] V. S. Miller. The weil pairing, and its efficient calculation. *J. Cryptol.*, 17(4):235–261, Sept. 2004.

[28] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In *VLDB*, 2006.

[29] H. Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, June 1984.

[30] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, 2000.

[31] M. L. Yiu, C. Jensen, X. Huang, and H. Lu. SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services. In *International Conference on Data Engineering (ICDE)*, pages 366–375, 2008.

APPENDIX

A. PRIMER ON HVE ENCRYPTION

HVE is built on top of a symmetrical bilinear map of composite order [4], which is a function $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ such that $\forall a, b \in G$ and $\forall u, v \in \mathbb{Z}$ it holds that $e(a^u, b^v) = e(a, b)^{uv}$. $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic multiplicative groups of composite order $N = P \cdot Q$ where P and Q are large primes of equal bit length. We denote by $\mathbb{G}_p$, $\mathbb{G}_q$ the subgroups of $\mathbb{G}$ of orders P and Q, respectively. Let l denote the HVE *width*, which is the bit length of the attribute, and consequently that of the search predicate. HVE consists of the following phases:

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

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

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

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

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

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

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

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.