GridSE: Towards Practical Secure Geographic Search via Prefix Symmetric Searchable Encryption

Ruoyang Guo
Stevens Institute of Technology

Jiarui Li
Stevens Institute of Technology

Shucheng Yu
Stevens Institute of Technology

Abstract

The proliferation of location-based services and applications has brought significant attention to data and location privacy. While general secure computation and privacy-enhancing techniques can partially address this problem, one outstanding challenge is to provide near latency-free search and compatibility with mainstream geographic search techniques, especially the Discrete Global Grid Systems (DGGS). This paper proposes a new construction, namely GridSE, for efficient and DGGS-compatible Secure Geographic Search (SGS) with both backward and forward privacy. We first formulate the notion of a semantic-secure primitive called *symmetric prefix predicate encryption* (SP^2E), for predicting whether or not a keyword contains a given prefix, and provide a construction. Then we extend SP^2E for dynamic *prefix symmetric searchable encryption* (pSSE), namely GridSE, which supports both backward and forward privacy. GridSE only uses lightweight primitives including cryptographic hash and XOR operations and is extremely efficient. Furthermore, we provide a generic pSSE framework that enables prefix search for traditional dynamic SSE that supports only full keyword search. Experimental results over real-world geographic databases of sizes (by the number of entries) from 10^3 to 10^7 and mainstream DGGS techniques show that GridSE achieves a speedup of $150\times$ - $5000\times$ on search latency and a saving of 99% on communication overhead as compared to the state-of-the-art. Interestingly, even compared to plaintext search, GridSE introduces only $1.4\times$ extra computational cost and $0.9\times$ additional communication cost. Source code of our scheme is available at https://github.com/rykieguo1771/GridSE-RAM.

1 INTRODUCTION

With the increasing deployment of Internet of Things (IoT) devices and the advent of next-generation wireless and mobile communications, the demand for Geographic Information Systems (GIS), particularly Location-Based Services (LBS), and consequently Geographic Searches (GS), has never been greater in various context-aware applications. To deliver accurate LBS services, GS apps usually need to collect fine-grained personal information such as users' real-time locations and interests. For privacy concerns, data stored in GS servers, which are usually hosted in the cloud, shall be well protected, e.g., via encryption, as a preemptive measure in light of data breaches. *Secure Geographic Search* (SGS) thus emerges as a promising data protection technique that supports GS over encrypted geographic data.

Existing GS systems, driven by various functionalities (not limited to searches), have been deeply rooted on a so-called *Discrete Global Grid* (DGG) [1] technique for structuring and indexing two-dimensional spatial data (or higher dimension extension). DGG uses hierarchical tessellation of grids to recursively partition the Earth's surfaces into geometrical cells which are then coded. A sequence of DGG cell codes describe a hierarchical partition with progressively finer resolution. As shown in Fig. 1, a cell code "$dr5r7$" represents a cell (sub-region) inside the grid of "$dr5r$" around the Statue of Liberty. DGG systems (DGGS) can also provide indexes/APIs to support various GIS/LBS services such as geographic search, distance comparison and spatial visualization. In real-world systems, DGGS technologies such as Niemeyer's Geohash [2], Google S2 [3] and Uber H3 [4] have been widely adopted.

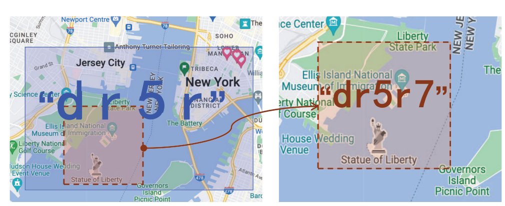


Figure 1: "$dr5r$" tags an area around the Statue of Liberty. Cell code "$dr5r7$" tags an area inside "$dr5r$".

With DGGS, a GS database is indexed by the hierarchical cell codes. A geographic search is essentially to find a prefix match between the indexed cell codes and the search request. To support SGS wherein both index and search request are encrypted, the problem can be formulated as the general problem of *secure prefix search*.

One outstanding challenge with the design of SGS, however, is to assure near latency-free instant responses for pleasant user experiences while compatible with existing GIS/LBS techniques especially DGGS. Theoretically, many advanced techniques, such as Fully Homomorphic Encryption (FHE) [5, 6] and Multi-Party Computation (MPC) [7], could be employed to realize SGS by performing prefix search over ciphertexts. However, current implementations of these techniques are still far from practical when real-time performances are of concern. For example, MPC requires multiple rounds of interactions between client and server for every single query. Current FHE-based secure search is relatively more efficient but still requires thousands of seconds per 10, 000 search tuples and a heavy communication cost [5]. On the other hand, emerging privacy-enhancing techniques such as Differential Privacy (DP) can be leveraged for *geo-indistinguishability* [8] and hence a more efficient SGS. However, such techniques are intrinsically lossy and will cause the so-called *lack-in-accuracy* issue in geographic services. One branch of privacy-preserving retrieval relies on anonymous technologies like *k-anonymity* [9], which are constrained by data distribution.

Aiming at secure search over encrypted data, searchable symmetric encryption (SSE) provides a generic but viable solution. However, constructing SGS from existing SSE is by no means trivial, especially considering the dynamics of GS databases (e.g., insertion and deletion of entries) and the privacy implications caused by such dynamics. Specifically, additional privacy, including *forward privacy* [10–13] and *backward privacy* [12, 14, 15], shall be considered for dynamic databases. Forward privacy is to ensure newly added data cannot be linked to previous queries while backward privacy further eliminates the link between a query and previously deleted data. These metrics and their definitions construct a strict framework for provable dynamic searchable security.

To our knowledge, the majority of existing SSE techniques focused on keyword search rather than prefix search and thus cannot support SGS, not to mention forward and backward privacy. Few works such as Moataz et. al. [16] enable substring search and provide a semantic-secure SSE but fail to address forward privacy or backward privacy. Moreover, their approach leverages Oblivious RAM (ORAM) which incurs a high communication complexity (quadratic logarithmic). For instance, considering a GS dataset of 10^5 entries, conducting a single search under this scheme needs at least hundreds of interactions which is far from practical for GIS/LBS applications. Another line of research [17–21] build a specialized index for SGS and all exhibit trade-offs between efficiency, security and false-positive rates.

In this paper, we construct an efficient, secure and DGGS-compatible SGS scheme that supports both forward privacy and backward privacy. To this end, we first formally formulate a novel cryptographic primitive *symmetric prefix predicate encryption* (SP^2E) for predicating a prefix, i.e., *evaluating whether a keyword contains a given prefix*, and provide a concrete construction. Based on SP^2E, we design a new cryptographic algorithm *prefix symmetric searchable encryption* (pSSE), namely GridSE, for secure geographic search that supports both forward and backward privacy. Interestingly, the construction of GridSE involves only cryptographic hash and XOR operations and is extremely light-weight. Regarding the search complexity, GridSE only introduces an additional cost linearly proportional to the number of DGGS cells, which is practically a constant in real systems and independent to the global GS database size. Thus, GridSE is highly applicable to large-scale GIS/LBS systems.

Our main contributions are summarized as follows:

- We construct an efficient, scalable and DGGS-compatible SGS scheme GridSE that supports both forward and backward privacy. Based on lightweight cryptographic primitives, GridSE is able to offer near real-time latency for large-scale geographic searches.

- We formulate and design a novel cryptographic primitive - symmetric prefix predicate encryption (SP^2E) - for predicating prefixes. We prove that SP^2E is secure under the random oracle model. Based on SP^2E, we construct a new prefix SSE primitive pSSE which is forward and backward secure. Both SP^2E and pSSE are generic primitives and can be of independent interests.

- We propose a generic dynamic pSSE framework based on the key idea of GridSE. It supports forward and backward privacy and is also compatible with traditional dynamic SSE that supports only full keyword search.

- Experiments on geographic databases with 10^3 - 10^7 entries and major DGGS systems (including Niemeyer's Geohash, Google S2 and Uber H3) show that GridSE achieves a speedup of $150\times$ - $5000\times$, respectively, in search latency and a saving of 99% in communication overhead as compared to the state-of-the-art [16]. Even compared with plaintext geographic search, GridSE only introduces $1.4\times$ additional computational cost and $0.9\times$ additional communication cost.

The rest of this paper is organized as follows: Section 2 presents the background. Section 3 lists related works which are followed by preliminaries in Section 4. The SP^2E primitive is introduced in Section 5. Section 6 presents our construction of GridSE. The experimental evaluation is shown in Section 7. We conclude this paper in Section 8.

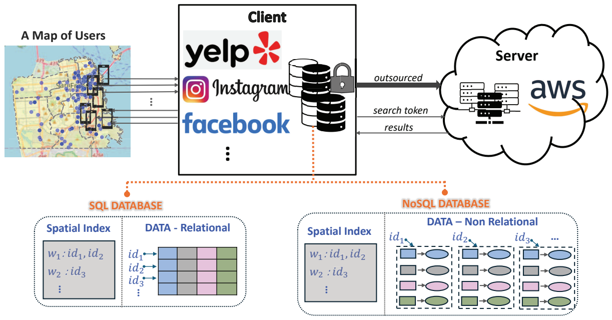


Figure 2: A geographic search framework

2 BACKGROUND

2.1 Geographic Search Framework

We consider a typical GIS/LBS application scenario with three major entities - *users*, *client* and *server*. A server (e.g., AWS) is a cloud provider for computing or storage services; a client is a GIS/LBS provider (e.g., yelp) that outsources its data (after encryption) to the server while keeping some local computation and storage capacity; users (e.g., mobile users) are consumers who make geographic search requests to the client to enjoy GIS/LBS services. We assume the client pre-computes encrypted indexes and uploads them to the server to assure search efficiency.

In real GIS/LBS systems, both SQL/relational and NoSQL/non-relational databases are widely adopted. To ensure compatibility, we consider secure geographic search scenarios with both types of databases. In SQL databases, data is organized in tables of columns (i.e., attributes) and rows (i.e., records). A record would be contributed by a user. NoSQL databases, on the other hand, store data as JSON documents which are composed of key-value pairs. For brevity, we use *block* to denote either a record or a JSON document since both of them can be indexed by a certain attribute. Each block is assigned a unique identifier, as shown in Fig. 2.

We logically divide a database into two pools, data and their indexes. Data (namely blocks) are indexed by their locations, and those sharing the same location are included under the same key. To avoid ambiguity, we call a key for indexing locations as *index-key*. For example, assume the index-key w_1 in Fig. 2 is associated with a certain geographic area on Earth (e.g., New York City). When searching for w_1, the data such as id_1, id_2 located in this area will be returned. In this work, we only focus on the location attribute (e.g., DGGS-compatible geographical cell codes) and how to retrieve an exact index-key over its encrypted form such that the queried encrypted blocks will be returned correctly, securely and efficiently.

2.2 Discrete Global Grid System

A Discrete Global Grid System (DGGS) [1] partitions the Earth areas into geometrical grids. Each grid can be further recursively partitioned into smaller regions which are called *cells*, with progressively finer resolution. For ease of calculation, each region or cell is represented by a point and assigned an identifiable string which we refer to as a *cell code*. Each cell can be associated with data objects (e.g., blocks as in Sec. 2.1) to facilitate geographical search. The recursive partition means that the grid is hierarchical, so are the cell codes. A longer cell code means a more precise geographical location and its prefix represents a broader area around it. Different DGGS systems may have different resolutions based on how and the granularity of the partition.

Widely adopted DGGS techniques include Niemeyer's Geohash [2], Google S2 [3] and Uber H3 [4], to name a few. They all decompose the globe into a hierarchy of regular geometric cells. For example, GeoHash partitions Earth into rectangle cells while Google S2 uses square cells and Uber H3 is based on hexagons. Geohash simply follows the hierarchical partition where the grid is recursively divided into smaller rectangular ones. S2 adopts so-called "roads" (for logical parent-child relationship between nodes) inside a cell such that points geographically close to each other are still close in the database. When a point is accessed via its index, its neighbours can also be accessed. These "roads" are coded in the prefix of a cell code. A similar strategy is adopted by H3. Taking the Statue of Liberty (Fig. 1) for instance, *Geohash* uses cell code "**dr5r7**" for the sub-area (the right sub-figure in Fig. 1) with a precision of $20km^2$. To search the similar area, *S2* uses "**b0fdf9d**", and it is **8a2a10** in *H3*. In Geohash, the cell code of the highest resolution of 6 cm^2 in that area is "**dr5r7**7*kkekp*9" with 12 levels (i.e., 12 letters) of precision. While in S2, it is "**b0fdf9d**0806*a*6787" for the highest resolution of $0.74cm^2$ with a maximum of 16 levels. H3 uses "**8a2a10**fffffffff" with a maximum of 15 levels for a precision of $8950cm^2$[1]. In essence, geographic search in DGGS is implemented through prefix search no matter what partitioning and cell coding methods are used.

2.3 Geographic Search Through Prefix Search

Consider an index-key w which is a cell code in a geographical database. Geographical search using another cell code w_p returns a match if and only if $w_p \sqsubseteq w$, i.e., *w_p is the prefix of w*. For example, when a tourist searches "best places to visit in New York", it will return a list of specific places such as "Empire State Building" and "Statue of Liberty". Since the query location "New York" (i.e., w_p) is broader than "Empire State Building" or "Statue of Liberty" (i.e., w that is used as an index-key in the geographical database), the

[1]This precision indicates the average hexagon area of H3 grids.

former is a prefix of the latter. Evaluating whether a search query is a prefix of an index-key is straightforward and can be illustrated as follows.

$$\boldsymbol{d} \rightarrow \boldsymbol{r} \rightarrow \mathbf{5} \rightarrow \boldsymbol{r} \rightarrow \mathbf{7} \rightarrow 7 \rightarrow \ldots \rightarrow k \rightarrow p \rightarrow 9$$

Figure 3: Prefix search for index-key "**dr5r7**7*kkekp*9"

Assume an index-key "**dr5r7**7*kkekp*9". Checking whether **dr5r7** is its prefix is simply to compare each character from left to right in the order of **d**, then **r**, ..., and then **7**, one by one, as shown in Fig 3. It shall be noted that in SGS where both search queries and index-keys (which are usually called "keywords") are encrypted, the same prefix search process as in Fig. 3 is performed but over encrypted characters.

3 RELATED WORK

Secure Geographic Search (SGS). The notion of Secure Geographic Search (SGS) was coined in 2014 by Ghinita and Rughinis [18] for searching encrypted geographic data. This work exploits Hidden Vector Encryption (HVE) to hierarchically encode and then encrypt data and queries to vectors, each of which is of high dimension. Thereafter, variant solutions [17, 19–23] have been proposed with improved efficiency. For instance, ref. [19] is constructed on a *domain-based* binary tree, and [20] on a Huffman encoding tree, both for faster retrieving of HVE [24] instances. Ref. [17] leverages Shen-Shi-Waters (SSW) encryption [25] to check and verify inner products over vectors converted from data and queries. However, the intrinsic complexities of HVE and SSW hinder the run-time efficiency despite of the theoretical *faster-than-linear* [17] and *logarithmic* [18–20] retrieve complexities, respectively. For retrieve efficiency, ref. [21] introduces a multi-level index mechanism based on bloom filters. However, this work also inherits the security-efficiency trade-off with bloom filters and offers a weaker security when high efficiency is preferred. Ref. [22] exhibits the similar issue and provides at most an equivalence of 17-bit security though only lightweight primitives such as cryptographic hash functions (based on the structure of [21]) and bloom filters are used. Besides, both [21] and [22] require one extra interaction for a single query. The above solutions consider a single server and follow the *single server-client* framework. Ref. [23] adopts two *non-colluding* servers but overlooks the leakage risk by simply clustering points by distance. Among these solutions, only [21] and [22] consider compatibility with the DGGS technologies. However, none of the existing solutions simultaneously satisfies the requirements of near real-time search latency, data dynamics, and forward/backward privacy.

SSE and Dynamic SSE. The problem of SSE was first formulated by Song et al. [26]. Since then, extensive research has been focused on various aspects including query expressiveness [27,28], search security with controlled leakage [29] and efficiency optimization [30, 31]. This family of SSE schemes allow leakage that is modeled under the notion of *access pattern* and *search pattern*, where the former reveals the actually retrieved items of a query, and the latter leaks previous queries related to the current one.

Stefanov et al. [32] first elaborated the notion of forward and backward privacy for so-called *dynamic SSE*, i.e., SSE that support data dynamics (i.e., data insertion and deletion). After that, a number of dynamic SSE schemes [10–13] were proposed but only supporting forward privacy, i.e, breaking the link between newly added data and previous queries. Zhang et. al. [33] further demonstrated a successful file-injection attack when forward privacy is not assured. The notion of backward privacy for dynamic SSE was first defined by Bost et al. [12], i.e., limiting what the server can learn between newly deleted data and queries afterwards. It regulates three different levels of leakages in the incremental (inclusive) order, from the least of Type-I to the most of Type-III. All these types allow the leakage of identifiers of documents that match the keyword w that is currently being searched and the time when they were inserted. Under this backward privacy framework, Bost et al. [12] and Ghareh Chamami et al. [14] proposed dynamic SSE constructions with all the three types of backward privacy, respectively. Sun et al. [34] proposed a Type-III backward privacy scheme. Recently, three Type-II schemes were proposed by Demertzis et al. [15, 35] and Sun et al. [36]. In addition, Hoang et al. [37] proposed a dynamic SSE framework particularly towards cloud data storage-as-a-service infrastructures. All these schemes focus on whole keyword search without supporting prefix search.

ORAM-TEE-based SSE. The leakage of *access pattern* has drawn increasing attention as discussed by Garg et. al [38]. Although this issue can be mitigated by Oblivious Random Access Machine (ORAM) [39] and its variants [40–42], deploying ORAM in the network environment, especially for searching data from remote cloud servers, incurs a significant bandwidth cost of at least poly-logarithmic complexity [43,44]. To address this challenge, Hoang et. al [45] combine ORAM with trusted execution environments (TEEs) to alleviate the bandwidth constraints. This approach leverages the trusted enclaves to store the oblivious structure and secret keys. Consequently, the decryption of the data during searches is executed within the enclave through I/O operations, rather than via communication with clients, reducing the bandwidth overhead by around 100× [45]. Another work [46] that adopts a similar methodology focuses on hiding the leakage caused by dynamic updates through utilizing the trusted components provided by TEE. Despite the great advancements, these works rely on a stronger security assumption on the TEE and only support search on the whole keywords.

Secure Substring Search (Prefix Search). Substring SSE was first studied by Shen et al. [47] in the context of static SSE. This construction is based on a suffix tree in which a large extent of the tree structure will be disclosed. Following this initial attempt, ref. [48] further reduces the storage cost of [47] and enhances its security. Leontiadis et. al. [49] then adopted Order-Preserving Encryption (OPE) for substring search. Subsequently, ref. [50] utilizes the SGX technology for substring search. The security of this approach hinges on the security of trusted hardware. Notably, all these constructions are for static SSE and leaks statistical information, e.g., the relation between suffix and prefix data. To support dynamic SSE, Moataz et. al. [51] utilize structure encryption and inner product predicate for substring search and updates. While accommodating updates, this method also leaks statistical information, such as the frequency of the same letters. In contrast, without compromising statistical leakage, ref. [16] utilizes a hierarchical ORAM tree structure to support substring search. Also, oblivious suffixes are organized by this structure to simultaneously support data update operations. However, this update introduces additional heavy bandwidth overhead due to the ORAM environment. In addition, Mainardi et al. [52] exclusively supports offline model for secure substring search. Other than these works, to our best knowledge, the research on the problem of secure substring or prefix search for dynamic SSE is very limited.

4 PRELIMINARIES

In this section, we define the basic cryptographic primitives used in this work, including the syntax and security definitions of dynamic prefix symmetric searchable encryption (pSSE) and forward/backward privacy.

4.1 Dynamic pSSE

Different from SSE, dynamic prefix symmetric searchable encryption (pSSE) extends the capabilities of SSE by enabling prefix search, alongside the inherent properties of SSE. Essentially, SSE can be viewed as a special case of pSSE where the prefix is the entire keyword. Similar to dynamic SSE, dynamic pSSE is pSSE that supports data dynamics such as insertion and deletion. A dynamic pSSE scheme $\Sigma = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ is comprised of one algorithm and two protocols:

$\mathsf{Setup}(1^\lambda) \to K, \sigma, EDB$: It takes as input the security parameter λ and outputs K, σ and EDB, where K is a secret key, σ is the client's local state, and EDB is the (empty) encrypted database that is to be sent to the server.

$\mathsf{Search}(K, \sigma, q; EDB) \to DB(w_p)$: It is a protocol for searching the database by the server. It gets inputs (K, σ, q) from a client and EDB (including index) from the server. The server outputs $DB(w_p)$ in the form of a result list I_{w_p}. The result would be empty if $w_p \not\sqsubseteq w$ for any index-key w. In this paper, we only consider search queries for a single prefix w_p of possibly multiple keywords (i.e., index-keys) w.

$\mathsf{Update}(K, \sigma, op, in; EDB) \to K, \sigma, EDB$: It is a protocol for inserting an entry to or deleting an entry from the database. It takes as inputs EDB from a server and (K, σ, op, in) from a client. Operation op can be add or del and the input in is composed of a block B and its identifier id. After the protocol, the input block B is added to or (logically) removed from EDB while its identifier id is added to or (logically) removed from its corresponding index. The protocol may modify K, σ and EDB as needed.

Above APIs follow the definition of [12] with some minor modifications. In this paper, we consider a database (SQL or NoSQL) composed of index and data, where the data is in the format of *blocks*. As discussed in Section 2.1, a block is either a record in SQL database or a JSON document in NoSQL database. In this work, EDB consists of two encrypted pools - the encrypted indexes and the encrypted blocks. We implicitly assume that after receiving the result identifiers $DB(w_p)$, the client performs an additional round to retrieve the actual blocks. For brevity, we omit this step in our construction.

CORRECTNESS: A dynamic pSSE scheme $\Sigma = (\mathsf{Setup}, \mathsf{Search}, \mathsf{Update})$ is correct if it always returns the correct results, i.e., indices, for each query. The definition aligns with the formal correctness definition of SSE in [29, 53].

SECURITY: The security of pSSE is captured by using a real-world versus ideal-world experimentation [29, 32, 53]. An intuition is that an adversary can not distinguish between the experiments REAL and IDEAL. The security model is parameterized by a leakage function $\mathcal{L} = \{\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Upt}\}$ which captures the information learned by an adversarial server. $\mathcal{L}^{Stp}, \mathcal{L}^{Srch}$ and $\mathcal{L}^{Upt}$ correspond to leakage during setup, search and updates in respective. Informally, a secure pSSE scheme with leakage $\mathcal{L}$ should reveal nothing about the database DB except for this explicit leakage.

***Definition* 4.1** (*Adaptive Security of Dynamic pSSE*)**.** A dynamic pSSE scheme Σ is $\mathcal{L}$-adaptively secure (with respect to leakage function $\mathcal{L}$) *iff* for all PPT adversary $\mathcal{A}$ that makes polynomial number of queries q, there exists a stateful PPT simulator Sim such that

$$|Pr[\mathrm{REAL}^{\Sigma}_{\mathcal{A}}(\lambda, q) = 1] - Pr[\mathrm{IDEAL}^{\Sigma}_{\mathcal{A},\mathsf{Sim}}(\lambda, q) = 1]| \le \mathrm{negl}(\lambda), \quad (1)$$

where $\mathrm{REAL}^{\Sigma}_{\mathcal{A}}(\lambda, q)$ and $\mathrm{IDEAL}^{\Sigma}_{\mathcal{A},\mathsf{Sim}}(\lambda, q)$ are defined as follows:

- $\mathrm{REAL}^{\Sigma}_{\mathcal{A}}(\lambda, q)$: The adversary $\mathcal{A}$ initially chooses a database DB and gets back EDB by calling $\mathsf{Setup}(1^\lambda)$. Then $\mathcal{A}$ is allowed to adaptively perform search (resp. update) queries with input q (resp. input (op, in))) and receives transcript generated by calling $\mathsf{Search}(K, \sigma, q; EDB)$ (resp. $\mathsf{Update}(K, \sigma, op, in; EDB)$). Given these real transcripts, the adversary $\mathcal{A}$ finally outputs a bit b.

- $\text{IDEAL}^{\Sigma}_{\mathcal{A},\mathsf{Sim}}(\lambda, q)$: The adversary $\mathcal{A}$ initially selects a database DB and gets back EDB generated by the simulator $\mathcal{S}(\mathcal{L}^{Stp}(DB))$. Then $\mathcal{A}$ is allowed to adaptively perform search (resp. update) queries with input q (resp. input (op, in))) and receives transcript generated by running $\mathcal{S}(\mathcal{L}^{Srch}(K, \sigma, q; EDB))$ (resp. $\mathcal{S}(\mathcal{L}^{Upt}(K, \sigma, op, in; EDB))$). Given these simulated transcripts, the adversary $\mathcal{A}$ finally outputs a bit b.

4.2 Forward and Backward Privacy of pSSE

Forward and backward privacy are two important properties of dynamic SSE that restrict what information is leaked by the Update query, so are they in dynamic pSSE. The definition of forward and backward privacy of SSE was first proposed in [32] and then formulated by Bost et al. [11, 12]. In their works, the definition only considers queries on keyword/document pairs, i.e., searching to find out whether a keyword belongs to a certain document. In this paper, we first formulate the definition of *forward and backward privacy* for a SQL (or NoSQL) database structures of index and data. A search is to find out the blocks that are related to an index-key w which is subject to a certain attribute. In brief, forward privacy ensures previous search queries do not reveal any information about the retrieved blocks to be updated. It hides whether an addition is about a new block or one that has been previously searched for. Backward privacy limits the information that the server can learn about blocks that are added previously but deleted later during searches related to them. It guarantees that the adversary cannot obtain extra information about the deleted blocks. For space limit, we include the full definitions of forward/backward privacy for keyword/block pairs in Appendix A, following similar definitions in [12].

4.3 Pseudorandom Function

Let $G : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a function defined from $\mathcal{X}$ to $\mathcal{Y}$. G is a secure pseudorandom function (PRF) if for all PPT adversaries $\mathcal{A}$, its maximum advantage as $Adv^{PRF}_{\mathcal{A},G}(\lambda) = |Pr[\mathcal{A}^{G(k,\cdot)}(1^\lambda) = 1] - Pr[\mathcal{A}^{F(\cdot)}(1^\lambda) = 1]|$ is negligible in λ, where F is a random function from $\mathcal{X}$ to $\mathcal{Y}$, and $k \xleftarrow{\$} \mathcal{K}$.

5 SYMMETRIC PREFIX PREDICATE ENCRYPTION

In this section, we present a novel cryptographic primitive, *symmetric prefix predicate encryption* (SP^2E). We begin with the building block *f-bit bounded symmetric encryption* (SE-f) followed by the syntax, security, and concrete construction of SP^2E.

5.1 f-bit Bounded Symmetric Encryption

Let λ be a security parameter and G denote a pseudorandom function with an output length $l(\lambda)$. An f-bit bounded symmetric encryption scheme SE-f with message space $\mathcal{M}$ can be described as a 3-tuple $(\mathsf{SE}\text{-}f.\mathsf{Gen}, \mathsf{SE}\text{-}f.\mathsf{Enc}, \mathsf{SE}\text{-}f.\mathsf{Dec})$. While the proposed scheme is derived from standard symmetric encryption (SE), the difference is in SE-f, only a f-bit interval is decrypted instead of the whole message. We formalize the definition of SE-f as follows:

$\mathsf{SE}\text{-}f.\mathsf{Gen}(1^\lambda) \to k_1, k_2$: On input a security parameter λ, it outputs two random secret keys k_1, k_2 uniformly sampled from key space $\mathcal{K}$.

$\mathsf{SE}\text{-}f.\mathsf{Enc}(k_1, \mathbf{m}) \to ct$: On input a secret key k_1 and message $\mathbf{m} \in \mathcal{M}$, it computes and outputs ciphertext ct as:

$$ct = G(k_1, \mathbf{m}) \oplus \mathbf{m}. \tag{2}$$

where $G(k_1, \mathbf{m})$ is a PRF output that XOR the message $\mathbf{m}$. Note that $|\mathbf{m}| = |G(k_1, \mathbf{m})|$, $ct \in \{0,1\}^{l(\lambda)}$.

$\mathsf{SE}\text{-}f.\mathsf{Dec}(k_1, k_2, ct) \to \mathbf{m}'$: On input k_1, k_2 and ct, it decrypts some chosen consecutive f bits among a length $l(\lambda)$ of the message $\mathbf{m}$ with the bit position starting from pos_1 to pos_2, where $0 \leq pos_1 < pos_2 < l(\lambda)$, $pos_2 - pos_1 = f$. The algorithm computes and outputs $\mathbf{m}'$ as:

$$\begin{aligned} \mathbf{m}' = ct \oplus & \\ & (G(k_2, \mathbf{m}).\mathsf{sub}(0, pos_1) \parallel G(k_1, \mathbf{m}).\mathsf{sub}(pos_1, pos_2) \parallel \\ & (G(k_2, \mathbf{m}).\mathsf{sub}(pos_2, l(\lambda)) \end{aligned} \tag{3}$$

where the method $ct.\mathsf{sub}(\mathsf{StartInx}, \mathsf{EndInx})$ extracts the bits from the StartInx-th position to the EndInx-th position in a "left-closure right-open" manner, e.g. for $ct = 100100$, $x.\mathsf{sub}(2, 4) = 01$.

CORRECTNESS:

$$\begin{aligned} & \mathbf{m}'.\mathsf{sub}(pos_1, pos_2) \\ &= \mathsf{SE}\text{-}f.\mathsf{Dec}(k_1, k_2, \mathsf{SE}\text{-}f.\mathsf{Enc}(k_1, \mathbf{m})).\mathsf{sub}(pos_1, pos_2) \\ &= (ct \oplus G(k_1, \mathbf{m})).\mathsf{sub}(pos_1, pos_2) \\ &= (G(k_1, \mathbf{m}) \oplus \mathbf{m} \oplus G(k_1, \mathbf{m})).\mathsf{sub}(pos_1, pos_2) \\ &= m.\mathsf{sub}(pos_1, pos_2). \end{aligned} \tag{4}$$

SECURITY: The security of SE-f is defined by an IND-CPA experiment presented in Fig. 4. The security definition of SE-f is similar to the standard indistinguishability definition of symmetric encryption (SE) except for decryption, where a segment of the message is decrypted in SE-f rather than a complete message.

***Definition* 5.1** (*Semantic Security of SE-f*)**.** A symmetric encryption scheme $\mathsf{SE}\text{-}f = (\mathsf{SE}\text{-}f.\mathsf{Gen}, \mathsf{SE}\text{-}f.\mathsf{Enc}, \mathsf{SE}\text{-}f.\mathsf{Dec})$ is IND-CPA secure if for any security parameter $\lambda \in \mathbb{N}$ and PPT (probabilistic polynomial time) adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible in λ. That is,

$$Adv^{\mathrm{IND-CPA}}_{\mathcal{A},\mathsf{SE}\text{-}f}(\lambda) = \left| Pr[\mathbf{Expt}^{\mathrm{IND-CPA}}_{\mathcal{A},\mathsf{SE}\text{-}f}(\lambda) = 1] - \frac{1}{2} \right| \leq negl(\lambda). \tag{5}$$

$\mathbf{Expt}^{\mathrm{IND-CPA}}_{\mathcal{A},\mathsf{SE}\text{-}f}(\lambda)$:

$b \xleftarrow{\$} \{0,1\}; k \xleftarrow{\$} \mathcal{K}$
$(\mathbf{m}_0, \mathbf{m}_1, st) \leftarrow \mathcal{A}'^{O_k^{\mathsf{Enc}}(\cdot)}(1^\lambda)\ s.t.$
$|\mathbf{m}_0| = |\mathbf{m}_1|$
$ct^* \leftarrow \mathsf{SE}\text{-}f.\mathsf{Enc}(k, \mathbf{m}_b)$
$b' \leftarrow \mathcal{A}^{O_k^{\mathsf{Enc}}(\cdot)}(st, ct^*)$
Return $(b' = b)$.

$O_k^{\mathsf{Enc}}(\mathbf{m})$:

$ct \leftarrow \mathsf{SE}\text{-}f.\mathsf{Enc}(k, \mathbf{m})$
Return ct.

Figure 4: IND-CPA security of SE-f

5.2 Syntax and Security of SP^{2}E

A symmetric prefix predicate encryption scheme SP^{2}E is defined over a keyword space Λ^t where each character of a keyword belongs to the alphabet Λ, and t is the maximum length of a keyword. SP^{2}E consists of a quintuple of PPT algorithms $(\mathsf{KeyGen}, \mathsf{PreEnc}, \mathsf{Enc}, \mathsf{TKGen}, \mathsf{PrefDec})$. The preliminary encryption algorithm PreEnc encodes the keyword as a fixed-length hash value which serves as a message, an input of the encryption algorithm Enc, to compute the ciphertext ct. The algorithm TKGen outputs a token, namely the encrypted prefix. The prefix decryption algorithm PrefDec identifies if the ciphertext ct contains a specific prefix without revealing the keyword nor message. The definition is formalized as follows:

$\mathsf{KeyGen}(1^\lambda, t) \rightarrow SK, MSK$: On input a security parameter λ and a maximum length t of a keyword, it generates a secret key SK for encryption and a master secret key MSK that is used as seed.

$\mathsf{PreEnc}(MSK, seq, w) \rightarrow \mathbf{m}$: On input MSK, a string keyword w and a number seq that indicates an identical sequence of w among keywords, the algorithm outputs a fixed-length hash value $\mathbf{m}$.

$\mathsf{Enc}(SK, \mathbf{m}) \rightarrow ct, \delta$: On input SK and $\mathbf{m}$, the algorithm outputs a ciphertext ct and a local parameter δ that is stored in the local state.

$\mathsf{TKGen}(SK, MSK, \delta, seq, w_p) \rightarrow k'$: On input secret keys SK, MSK, a local parameter δ, a number seq and a prefix w_p, it outputs a token k'.

$\mathsf{PrefDec}(k', ct) \rightarrow \perp$ or $w_p \sqsubseteq w$: On input a token k' and ciphertext ct, it outputs $\perp$ or $w_p \sqsubseteq w$.

CORRECTNESS: For security parameter λ, integer $t \in \mathbb{N}^*$, $(SK, MSK) \leftarrow \mathsf{KeyGen}(1^\lambda, t)$ and a keyword $w \in \Lambda^t$, SP^2E is correct if

$$Pr\begin{bmatrix} \mathsf{PrefDec}(k', ct) & ct, \delta \leftarrow \mathsf{Enc}(SK, \mathsf{PreEnc}(MSK, seq, w)) \\ = \quad : & k' \leftarrow \mathsf{TKGen}(SK, MSK, \delta, seq, w_p), \\ w_p \sqsubseteq w & w_p \sqsubseteq w \end{bmatrix} = 1. \tag{6}$$

SECURITY: The security of SP^{2}E is defined by two experiments, an IND-CPA experiment $\mathbf{Expt}^{\mathrm{IND-CPA}}_{\mathcal{A},\mathrm{SP^2E}.\mathsf{Enc}}(\lambda)$ for encryption and an IND-f-CPA experiment $\mathbf{Expt}^{\mathrm{IND-}f\mathrm{-CPA}}_{\mathcal{A},\mathrm{SP^2E}.\mathsf{TKGen}}(\lambda)$ for token generation. The security definition of IND-f-CPA is similar to the standard IND-CPA indistinguishability definition except for the specific f bits. Concretely speaking, a PPT adversary has the ability to access and query both oracles $O^{\mathsf{Enc}}_{SK}(w)$ and $O^{\mathsf{TKGen}}_{SK,MSK,\delta}(w_p)$ with unlimited times after the adversary submits an input w (or w_p). The details of experiments are presented in Fig. 10 in Appendix B. The formal definition of encryption and token security of SP^{2}E is presented below.

***Definition* 5.2** (*Encryption Security of SP^{2}E*)**.** A symmetric prefix predicate encryption scheme SP^{2}E is IND-CPA encryption secure, if for all PPT adversary $\mathcal{A}$, the advantage pf $\mathcal{A}$ is negligible in λ. That is:

$$\begin{aligned} Adv^{\mathrm{IND\text{-}CPA}}_{\mathcal{A},\mathrm{SP^2E}.\mathsf{Enc}}(\lambda) &= \left| Pr[\mathbf{Expt}^{\mathrm{IND\text{-}CPA}}_{\mathcal{A},\mathrm{SP^2E}.\mathsf{Enc}}(\lambda) = 1] - \frac{1}{2} \right| \\ &\leq negl(\lambda). \end{aligned} \tag{7}$$

***Definition* 5.3** (*Token Security of SP^{2}E*)**.** A symmetric prefix predicate encryption scheme SP^{2}E is IND-f-CPA token secure, if for all PPT adversary $\mathcal{A}$, the advantage pf $\mathcal{A}$ is negligible in λ. That is:

$$\begin{aligned} Adv^{\mathrm{IND\text{-}}f\mathrm{\text{-}CPA}}_{\mathcal{A},\mathrm{SP^2E}.\mathsf{TKGen}}(\lambda) &= \left| Pr[\mathbf{Expt}^{\mathrm{IND\text{-}}f\mathrm{\text{-}CPA}}_{\mathcal{A},\mathrm{SP^2E}.\mathsf{TKGen}}(\lambda) = 1] - \frac{1}{2} \right| \\ &\leq negl(\lambda). \end{aligned} \tag{8}$$

According to the encryption and token security defined in Def. 5.2& 5.3, we further define the *adaptive security* for SP^{2}E as follows:

ADAPTIVE SECURITY: We define the adaptive security for SP^{2}E scheme since a PPT adversary $\mathcal{A}$ is allowed to submit the chosen challenged keyword strings w_0, w_1 (or prefixes w_{p_0}, w_{p_1}) adaptively at any point in time while the security is guaranteed.

***Definition* 5.4** (*Adaptive Security of SP^{2}E*)**.** A symmetric prefix predicate encryption scheme SP^{2}E is IND-f-CPA secure, if for all PPT adversary $\mathcal{A}$, it has at a most negligible advantage

$$\begin{aligned} Adv^{\mathrm{IND\text{-}}f\mathrm{\text{-}CPA}}_{\mathcal{A},\mathrm{SP^2E}}(\lambda) = \mathsf{MAX}\Big(&|Pr[\mathbf{Expt}^{\mathrm{IND\text{-}CPA}}_{\mathcal{A},\mathrm{SP^2E}.\mathsf{Enc}}(\lambda) = 1] - \frac{1}{2}|, \\ &|Pr[\mathbf{Expt}^{\mathrm{IND\text{-}}f\mathrm{\text{-}CPA}}_{\mathcal{A},\mathrm{SP^2E}.\mathsf{TKGen}}(\lambda) = 1] - \frac{1}{2}|\Big) \\ \leq negl(\lambda). \end{aligned} \tag{9}$$

5.3 Bit-wise Prefix Recognition SP^{2}E from PRF

In this part, we present the details of SP^{2}E based on a pseudorandom function (PRF) $G : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$, a cryptographic hash function $H : \mathcal{K} \times \mathbb{N}^* \rightarrow \mathcal{K}$ and an f-bit bounded symmetric encryption scheme $\mathsf{SE}\text{-}f = (\mathsf{SE}\text{-}f.\mathsf{Gen}, \mathsf{SE}\text{-}f.\mathsf{Enc}, \mathsf{SE}\text{-}f.\mathsf{Dec})$. Specifically, the proposed scheme $\mathsf{SP^2E} = (\mathsf{KeyGen}, \mathsf{PreEnc}, \mathsf{Enc}, \mathsf{TKGen}, \mathsf{PrefDec})$ is constructed as follows:

$\mathsf{KeyGen}(1^\lambda, t) \to SK, MSK$: On input a security parameter λ and a parameter t that denotes the maximum length of a keyword, it uniformly samples secret keys $sk_1, sk_2, sk_{c_1}, \ldots, sk_{c_{|\Lambda|}}$ at random from the key space $\mathcal{K}$. Each secret key sk_{c_i} uniquely associates with a character c_i in the textual alphabet Λ, for $1 \le i \le |\Lambda|$, and $|\Lambda|$ is the total number of distinct characters. A keyword is a string and all of its characters are from Λ. This algorithm specifies a parameter f for indicating the number of "certain bits" that are indicators of a queried result. f is subject to $0 < f \le \left\lfloor \frac{l(\lambda)}{t-1} \right\rfloor$. For simplicity, we denote the outputs with $SK = \{sk_1, sk_2\}$, $MSK = \{sk_{c_1}, \ldots, sk_{c_{|\Lambda|}}, f\}$.

$\mathsf{PreEnc}(MSK, seq, w) \to \mathbf{m}$: It takes as input MSK, a keyword w and a number seq that is the sequence of this w among (distinct) keywords. It outputs the fixed-length hash value $\mathbf{m}$ computed by following steps:

(1) Compute the length of a keyword w, $|w|$, i.e., the total number of its characters.

(2) Pick up the associated key sk_{w_i} from MSK for each character of w in sequence. For example, if the i-th character is a textual letter $'a'$, then its associated key is $sk_{'a'}$, namely sk_{w_i} refers to $sk_{'a'}$ in this case.

(3) Output $\mathbf{m}$ as

$$\mathbf{m} = \bigoplus_{i=1}^{|w|} \left(0^{(i-1)f} \parallel H(sk_{w_i} \parallel seq \parallel i)\right).\mathsf{sub}(0, l(\lambda)). \quad (10)$$

$\mathsf{Enc}(SK, \mathbf{m}) \to ct, \delta$: On input SK and $\mathbf{m}$, by using the secret key sk_1 in SK, it computes the ciphertext as

$$ct = \mathsf{SE}\text{-}f.\mathsf{Enc}(sk_1, \mathbf{m}) = \mathbf{m} \oplus G(sk_1, \mathbf{m}), \quad (11)$$

When the encryption is invoked, $\delta = G(sk_1, \mathbf{m})$ must be recorded in the local state. Note that δ is not public. Instead, it is secretly held by the entity executing this algorithm, usually the client.

$\mathsf{TKGen}(SK, MSK, \delta, seq, w_p) \to k'$: On input SK, MSK, a local parameter δ, an identifier seq and a queried prefix w_p, it outputs a token k' that will be used to evaluate if the message of a ciphertext ct contains prefix w_p. With both secret keys sk_1 and sk_2 in SK, the algorithm computes k' as:

(1) Calculate the length of the prefix w_p, $|w_p|$.

(2) Pick up the associated key $sk_{w_{p_i}}$ from MSK in sequence for each character of w_p.

(3) Set $pos_1 = (|w_p| - 1) \cdot f$, $pos_2 = |w_p| \cdot f$, and we have $pos_2 - pos_1 = f$. The parameters pos_1 and pos_2 are the respective start and end bit-position of an f-bit segment.

(4) Output k' as

$$\begin{aligned} k' = & \bigoplus_{i=1}^{|w_p|} \left(0^{(i-1)f} \parallel H(sk_{w_{p_i}} \parallel seq \parallel i)\right).\mathsf{sub}(0, l(\lambda)) \\ & \oplus \big(G(sk_2, seq).\mathsf{sub}(0, pos_1) \parallel \delta.\mathsf{sub}(pos_1, pos_2) \parallel \\ & \quad G(sk_2, seq).\mathsf{sub}(pos_2, l(\lambda))\big). \end{aligned} \quad (12)$$

$\mathsf{PrefDec}(k', ct) \to \perp$ or $w_p \sqsubseteq w$ [2]: On input a token k' and a ciphertext ct, it calculates that

$$r' = k' \oplus ct, \text{ and } 0^f \stackrel{?}{=} r'.\mathsf{sub}(pos_1, pos_2) \quad (13)$$

The algorithm evaluates whether there are consecutive f bits valued with zero located in the segment $[pos_1, pos_2)$ of the result r'. We denote these f bits with 0^f. In brief, 0^f is an indicator for a keyword. By verifying $0^f \stackrel{?}{=} r'.\mathsf{sub}(pos_1, pos_2)$, the algorithm $\mathsf{PrefDec}$ can identify whether the the message m contains the prefix w_p. The algorithm outputs $\perp$ if w_p is not a prefix of m. Otherwise, it outputs $w_p \sqsubseteq w$.

CORRECTNESS: The correctness of $\mathsf{SP^2E}$ is derived from that of PRF G and the symmetric encryption $\mathsf{SE}\text{-}f$. That is

"$w_p \sqsubseteq w$: w_p is a prefix of w" or
"$w_p \not\sqsubseteq w$: w_p is not a prefix of w".

In the first case, the bits valued with zero (i.e., 0^f) in the output of $\mathsf{SP^2E}$ decryption should be revealed such that a keyword w having the queried prefix w_p can be identified correctly. To validate this, we expand the output of $\mathsf{PrefDec}$, namely $k' \oplus ct$. As shown in Fig. 11, If w_p is a prefix of the keyword, the f bits between pos_1 and pos_2 should all be zero. As for the latter case of $w_p \not\sqsubseteq w$, the output of $\mathsf{SP^2E}$ decryption should be a completely indistinguishable value without any consecutive bits valued at zero visible. We validate this with more details and examples in Fig. 12 in Appendix D.

EXAMPLE ILLUSTRATION: We illustrate each algorithm of $\mathsf{SP^2E}$ with the example of the Statue of Liberty. Consider a DGGS code w = "**dr5r77**" and a queried prefix w_p = "**dr5r7**". With all the required parameters of KeyGen, PreEnc computes the fixed-length hash value $\mathbf{m}$ according to the inputs – a keyword w and its sequence seq. A fixed-length string is obtained by XORing several hash substrings that correspond to the characters of w (i.e., "$dr5r77$"), as depicted by a grey rectangular box in Fig. 5 (a). The encryption algorithm Enc generates the ciphertext ct by XORing $\mathbf{m}$ with a mask δ which is generated uniformly at random and stored locally, as shown in Fig. 5 (b).

TKGen outputs an encrypted search token k' for the query prefix w_p (i.e., "$dr5r7$"). As shown in Fig. 5 (c), k' is computed by XORing the hash substrings with a pseudorandom

[2] This decryption here is logically equivalent to the decryption $\mathsf{SE}\text{-}f.\mathsf{Dec}$ $(sk_1, sk_2, ct) \to \mathbf{m}$ in Sec. 5.1. The message carried by a token k' corresponds to the mentioned $\mathbf{m}$, that is $\bigoplus_{i=1}^{|w_p|} \left(0^{(i-1)f} \parallel H(sk_{w_{p_i}} \parallel seq \parallel i)\right).\mathsf{sub}(0, l(\lambda))$.

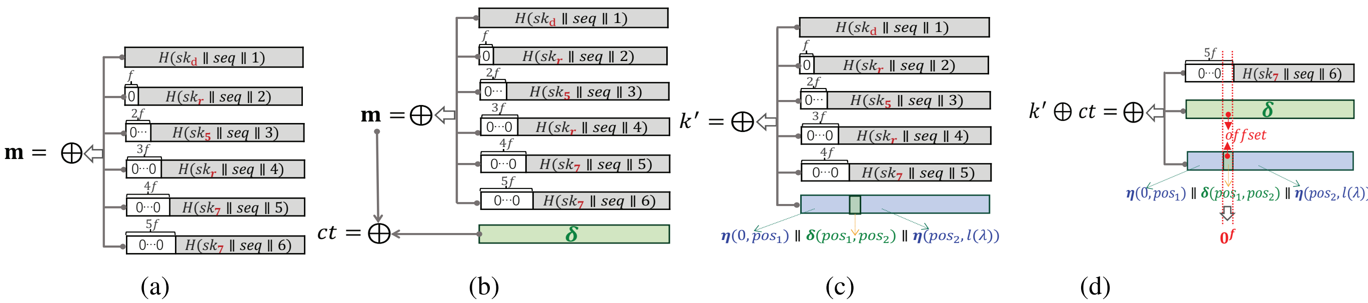


Figure 5: (a) $\mathbf{m} \leftarrow \mathsf{PreEnc}(MSK, seq, w)$ for $w =$ "**dr5r77**". (b) $(ct, \delta) \leftarrow \mathsf{Enc}(SK, \mathbf{m})$, where $\delta = G(sk_1, seq)$. (c) $k' \leftarrow \mathsf{TKGen}(SK, MSK, \delta, seq, w_p)$ for $w_p =$ "**dr5r7**", where $\eta = G(sk_2, seq)$. (d) The PrefDec evaluation on ct for this w and w_p.

string which is the concatenation of pseudo-random value η and a local secret δ. The hash substrings are associated sequentially with the characters of w_p. If SHA-256 is chosen for producing the hash substrings, for example, the output length $l(\lambda) = 256$.

With the input token k' and a ciphertext ct, the prefix decryption algorithm PrefDec examines whether the f bits in the output (i.e., $k' \oplus ct$) at positions $[pos_1, pos_2)$ are all zeros. All zeros mean a keyword w matches a given prefix w_p. As shown in Fig. 5 (d), bits $[4f, 5f)$ of $k' \oplus ct$ are all zeros, which occurs due to the two same values $\delta.\mathsf{sub}(pos_1, pos_2)$[3] presented in k' and ct, respectively.

5.4 Security Analysis

In this part, we show the proposed SP^{2}E of being proven IND-f-CPA secure in the random oracle model, under the security of cryptographic hash function G, PRF F and f-bit bounded symmetric encryption SE-f.

Theorem 5.5 (Adaptive Security). *If $G : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ is a secure PRF, H is a secure cryptographic hash function, H is modeled with a random oracle, and SE-f is an IND-CPA secure symmetric encryption, then the proposed SP^{2}E is IND-f-CPA secure in the random oracle model.*

Proof Sketch: We prove the security of our SP^{2}E scheme through two parts, encryption security defined in Def. 5.2 and token security defined in Def. 5.3, for which we give concrete proofs in respective via a series of games. Based on both encryption and token security, the adaptive security of SP^{2}E in Def. 5.4 holds. To complete the proof, we consider a sequence of hybrid games that differ from each other in the *challenge pre-encryption ciphertext* and *challenge token*. The first game in the sequence corresponds to the real IND-f-CPA game for SP^{2}E while the last hybrid one is the standard IND-CPA game for the f-bit bounded symmetric encryption SE-f. The overall advantage of $\mathcal{A}$ changes only by a negligible amount between each successive hybrid game, and is bounded by the advantage gained in IND-f-CPA game for token generation, which is negligible, thereby establishing Thm 5.5. Due to the limited space, the complete proof is provided in the extended version of this paper, which will be available on Arxiv.

6 GridSE: Dynamic pSSE for Secure Geographic Search from SP^{2}E

In this section, we present GridSE, a dynamic prefix symmetric searchable encryption (pSSE) based on symmetric prefix predicate encryption (SP^{2}E) for secure geographic search.

6.1 Construction Details

GridSE follows the standard APIs of a standard dynamic pSSE scheme as discussed in Sec 4.1. By adapting the definitions to a generic SQL or NoSQL database, we implement a practical dynamic pSSE system for fast secure geographic search over DGGS cells. Recall that the database is typically organized by blocks, i.e., files/records. To support forward and backward privacy, GridSE encrypts triplets of (w, id, op) into two ciphertext components which are organized as key-value pairs in a dictionary. The key stored in the dictionary is the ciphertext of *index-key* w, namely the address which indicates whether w is queried by a specific prefix during searches. The value in the dictionary is a list associated with w, consisting of one or multiple encrypted blocks (id, op). The encryption of (id, op) is based on a mask generated by PRF and encapsulates an update counter $updt_{cnt}$ that stores the update times occurred on blocks under index-key w as widely adopted for dynamic SSE. Note that data operations are indicated using $op = add/del$, and any deletion operation is logically recorded instead of executing a physical deletion action.

[3] In Fig. 5 (c)-(d) we omit "sub" for simplicity, i.e., $\delta(pos_1, pos_2) = \delta.\mathsf{sub}(pos_1, pos_2)$, $\eta(0, pos_1) = \eta.\mathsf{sub}(0, pos_1)$, etc.

However, a ciphertext of (id, op) generated only based on $updt_{cnt}$ is not enough to achieve forward and backward privacy for dynamic pSSE. The blocks (i.e., (id, op)) associated with different index-keys but with the same $updt_{cnt}$ will produce the same masks. To solve this issue, we introduce an additional parameter, the sequence of index-keys *seq* that helps to "tag" those blocks. In this way, each block is encrypted based on a unique mask and every update operation accessing the encrypted blocks in GridSE appears randomly to the server. Hence, the server cannot distinguish between insertion and deletion, which achieves backward privacy. Since each block is a separate triplet, the server cannot distinguish a new keyword in a newly added block from a keyword contained in the previously searched block, which achieves forward privacy. To complete a search, after receiving the index-keys for a prefix, the client can generate all masks for correctly decrypting each block. Notice that each index-key encrypted by SP^{2}E corresponds to a DGGS cell code and can identify the queried cells for the given prefix by the function SP^{2}E.PrefDec. GridSE hence enables efficient search while supporting forward and backward privacy. Detailed construction of GridSE is as follows:

$\mathsf{Setup}(1^\lambda) \rightarrow K_\Sigma, \sigma, EDB$: On input a security parameter λ, the client generates a local state σ and an empty encrypted database EDB. It obtains secret keys from sampling $K \xleftarrow{\$} \{0,1\}^\lambda$ and secret keys SK, MSK by invoking the Setup algorithm of SP^{2}E. Next, the client initiates an empty list $\hat{\delta}$ and two maps ($\mathbf{InxDict}, \mathbf{UpdtCnt}$). $\hat{\delta}$ and **UpdtCnt** are locally maintained while **InxDict** is sent to server. At last, client outputs $K_\Sigma = (K, SK, MSK), \sigma = \{\mathbf{UpdtCnt}, \hat{\delta}\}$ and EDB.

$\mathsf{Update}(K_\Sigma, \sigma, op = add/del, in = (w, id); EDB) \rightarrow K_\Sigma, \sigma, EDB$: In the update procedure, the client receives an index-key w, an identifier id and operation $op = add/del$. An input (add, w, id) means *"add a block id in EDB and an entry in **InxDict** for this block that has an index-key w"*. The client has access to the keys K_Σ, the local state **UpdtCnt** and $\hat{\delta}$, where $\hat{\delta}$ keeps the parameter δ that is produced once invoking SP^{2}E.Enc, and **UpdtCnt** stores for each distinct index-key w a counter that denotes how many updates have taken place on blocks in relation to w. First, the client checks whether $\mathbf{UpdtCnt}[w]$ has been initialized or not. If not, he sets the counter value of w to 0 (lines 1-4). Here, **UpdtCnt** is the size of current sequence of index-keys which indicates how many index-keys are currently in the dictionary. It increases by 1 once a new distinct index-key is produced with updates. Next, the client produces a key-value pair $(addr, \delta)$ using the algorithm SP^{2}E.Enc by encrypting a given index-key and its current sequence. δ is kept local. Notice that if a new but repeated index-key w is associated with an update after it has been counted in **UpdtCnt**, the newly produced address *addr* associated with this w will be the same as the previous one due to the same sequence and keys as the input (lines 5-7). The client then runs the

---

Algorithm 1: GridSE Σ : $\mathsf{Setup}(1^\lambda, DB)$

1: $(EDB, \sigma) \leftarrow \Sigma.\mathsf{Setup}(1^\lambda)$
2: $K \xleftarrow{\$} \{0,1\}^\lambda$, $SK, MSK \leftarrow \mathsf{SP^2E.Setup}(1^\lambda)$
3: $\hat{\delta} \leftarrow$ empty list
4: $\mathbf{UpdtCnt}, \mathbf{InxDict} \leftarrow$ empty linked map
5: $\sigma \leftarrow \mathbf{UpdtCnt}, \hat{\delta}$
6: Return $(K, SK, MSK), (\mathbf{UpdtCnt}, \hat{\delta}), EDB$

PRF G with key K and computes $G(K, (seq, \mathbf{UpdtCnt}[w]))$ that is XORed with the message $(id \parallel op)$, and this result becomes an encrypted value *val* (line 8). The pair $(addr, val)$ is sent to the server which stores them by appending *val* into $\mathbf{InxDict}[addr]$ or a newly produced $\mathbf{InxDict}[addr]$ (lines 9-13).

$\mathsf{Search}(K_\Sigma, \sigma, w_p; EDB) \rightarrow \mathrm{I_{w_p}}$: While searching for the index-keys with a certain prefix w_p, the client first generates a list of tokens $TList$. This token generation is submitted to the respective SP^{2}E layer, namely SP^{2}E.TKGen. $TList$ is then sent to the server (lines 1-6). Recall that the corresponding entries to this token are stored in **InxDict** on the server, which enables the server to evaluate these entries with the SP^{2}E function SP^{2}E.PrefDec. If SP^{2}E.PrefDec indicates *"w_p is a prefix for the current index-key w"*, the server obtains the associated key-value pairs of w from the index and puts the value and its sequence into R_{w_p}. Specifically, The server retrieves the key-value pairs matching the token $TList$ in **InxDict** and then sends them back to the client (lines 7-17). Then, the client decrypts the returned result R_{w_p} which are composed of pseudorandom values. These values generated via PRF G during the previous updates for w are recovered to the original plaintext since G is a deterministic function. Concretely, the client decrypts the received dictionary R_{w_p} by calculating the PRF values $G(K, (seq, i))$ and XORing them with each element of *valList* in sequence for $i = 1, ..valList.size$, where *seq* is the sequence of the key-value pair in which the current *valList* locates in **InxDict** (lines 18-25). In the end, the client returns a search result I_{w_p} containing all requested block identifiers which correspond to the index-keys of a prefix w_p.

***Theorem* 6.1.** *Assuming F is a secure PRF and SP^{2}E is IND-f-CPA secure, GridSE is an adaptively-secure dynamic pSSE scheme of forward and Type-II backward privacy with* $\mathcal{L}^{Updt}(op, w, id) = \perp$ *and* $\mathcal{L}^{Srch}(w_p) = (\mathsf{TimeDB}(w_p), \mathsf{Update}(w_p))$.

PROOF SKETCH: We prove the adaptive security of GridSE by constructing a sequence of games and show that the advantage of a PPT adversary against our protocol is negligible. Intuitively, the transcript $(addr, val)$ sent to the server is indistinguishable from uniformly random samples, which is guaranteed by the construction. Notice that the value *val* is computed by XORing the block/operation tuple $(id || op)$

Algorithm 2: GridSE Σ : Update $(K_\Sigma, \sigma, op = add/del, in = (w, id); EDB)$

```
Client:
  if UpdtCnt[w] is NULL then
      UpdtCnt[w] = 0
  end if
  UpdtCnt[w]++
  seq = UpdtCnt.getMapSize()
  addr, δ = SP²E.Enc(SK, SP²E.PreEnc(MSK, seq, w))
  σ ← δ
  val = (id || op) ⊕ G(K, (seq, UpdtCnt[w]))
  Returns (addr, val)
Server:
  if InxDict.containsKey(addr) then
      Set InxDict[addr] ∪ val
  else
      Set InxDict[addr] = val
  end if
```

Algorithm 3: GridSE Σ : Search $(K_\Sigma, \sigma, w_p; EDB)$

```
Client:
  TList ← {}
  for i = 1 to UpdtCnt.size do
      T_i = SP²E.TKGen(SK, MSK, δ_i, i, w_p)
      TList = TList ∪ {T_i}
  end for
  Return TList
Server:
  R_{w_p} ← empty linked map
  if InxDict.size ≠ TList.size then
      Abort
  end if
  for i = 1 to InxDict.size do
      (addr, valList) = InxDict.getEntry(i)
      if SP²E.PrefDec(TList[i], addr) == 0^f then
          Set R_{w_p}[i] = valList
      end if
  end for
  Return R_{w_p}
Client:
  I_{w_p} = {}
  while R_{w_p}.hasNext() do
      (seq, valList) = R_{w_p}.getNext()
      for i = 1 to valList.size do
          (id||op) = valList[i] ⊕ G(K, (seq, i))
      end for
      I_{w_p} = I_{w_p} ∪ (id||op)
  end while
```

with $G(K, (seq, \mathbf{UpdtCnt}[w]))$ generated by a pseudorandom function G. In this way, the server cannot distinguish whether an update is insertion or deletion, which means that the update leakage is thwarted. Regarding backward privacy, notice that when performing a search on prefix w_p, the server can retrieve the corresponding entries in **InxDict** with their sequences. Since these entries are encrypted by SP^{2}E and an additional pseudorandom function, the server can learn nothing except the time when each update operation takes place for w_p. This means GridSE provides Type-II backward privacy as defined in Def. A. The complete proof is omitted in this paper and will be available in the extended version in Arxiv.

Remark for Efficiency. We first introduce the notations for analyzing complexity. d_w denotes the number of updates on the distinct index-keys in **InxDict**. a_{w_p} represents the number of updates on a given prefix w_p, which is equivalent to the number of blocks contained in the result I_{w_p}. We show the computation, storage and communication complexity of GridSE in Table 1 for the reference of the following empirical evaluations.

A search comprises a sequence of operations such as generating tokens in the client, sweeping index-keys in the server and decrypting in the client. Particularly, it requires d_w evaluations of **InxDict** in the server, and d_w SP^{2}E.TKGen invocations and a_{w_p} PRF instances for decryption in the client. Recall that a SP^{2}E.TKGen invocation introduces a PRF and several hash instances while a SP^{2}E.PrefDec invocation requires a single XOR operation. Both of these invocations are all evaluated with constant. Thus, the overall complexity of a search operation is $O(a_{w_p} + d_w)$. Regarding the communication cost of a search, it is $O(a_{w_p})$ as the returned result size in a list format is asymptotic a_{w_p}. On the other hand, an update requires $O(1)$ for the communication overhead due to that the client only submits a single entry to the server. After N updates have taken place, regarding the client keeps a local list $\hat{\delta}$ of size d_w, the local storage is merely $O(d_w)$. Finally, GridSE takes an extra roundtrip to fetch the block identifiers for w_p if the actual blocks are necessary to be retrieved.

Recall that an index-key is associated exclusively with a DGGS cell. The maximum value of d_w does not exceed the number of DGGS cells, which is a constant for a given geographical region. A constant-constrained d_w implies a high search efficiency and tolerance of frequent updates in GridSE. This performance advantage is ensured by two properties, the fixed pattern of cell partitioning and the fixed resolution of a cell in DGGS. For instance, given a database located within the United States with a size up to 10^5, 10^6, or 10^7, the number of DGGS cells is stabilized at around 1615 with each cell in the $0.01km^2$ scale.

Cleaning Deleted Items. Allowing the size of EDB to grow with each update including deletions is a common strategy in building dynamic SSE [14, 35]. This simplifies the deletions while reaching forward and backward privacy. GridSE can mitigate this growth via a periodic "clean-up" operation that has been adopted in previous schemes [14]. The "clean-up" operation involves the client to remove the deleted identifiers in $valList$ when he decrypts a search result R_{w_p}. The remaining identifiers are then re-encrypted and sent

Setup

- Same as **Algorithm** 1. Client returns (K, SK, MSK), $(\mathbf{UpdtCnt}, \hat{\delta}), EDB$.

Update

- Client computes $< addr, val >$ and sends it to server, but maintain δ at local. These values are computed as:
  - $addr, \delta = \mathsf{SP^2E.Enc}(SK, \mathsf{SP^2E.PreEnc}(MSK, seq, w))$,
  - $val \leftarrow Forward/Backward.Enc_{seq,\mathbf{UpdtCnt}[w]}(id, op)$,

  where $seq =$ *(current)* **UpdtCnt**.size.
- Server stores $< addr, val >$ in a linked map.

Search

- Client generates token:

  $tk = \mathsf{SP^2E.TKGen}(SK, MSK, \delta_i, i, w_p)$ for $i = 1, ..., d_w$.
- Server checks whether an index-key matches a prefix or not:

  $\mathsf{SP^2E.PrefDec}(tk, addr) == 0^f$,

  If yes, obtains all associated vals related to $addr$, and sends them back to client.
- Client decrypts vals.

Figure 6: A generic dynamic pSSE framework with forward/backward privacy

back to the server. However, the deterministic encryption over identifiers with the repeated input to PRFs would lead to "identical" ciphertexts in *valList* which fail to protect privacy. "Identical" means the same ciphertexts as the previous ones. Hence, as a countermeasure, we add a new counter as input into the PRFs, which increments with every search cycle. To guarantee the correct token generation and decryption for subsequent searches, an additional counter map **IncreCnt** is required to be maintained locally. This design retains the same performance and backward privacy as GridSE. We omit the details of this part in this paper due to the space limit.

6.2 Towards Generic Dynamic pSSE

The key idea of GridSE to achieve forward/backward private dynamic pSSE is its utilization of SP^{2}E as a building block to support prefix search and its creation of a key-value dictionary based on the encrypted triplets (w, id, op). These techniques are also compatible with traditional dynamic SSE schemes that support only full keyword search. In this section, we provide a generic dynamic pSSE framework (Fig. 6) by leveraging the techniques in GridSE to enable prefix search capabilities for forward/backward private dynamic keyword SSE. In this framework, one vital requirement is that the keywords with the same prefix need to be grouped and associated with the same index-key in the key-value dictionary. The framework consists of two main steps. First, when searching for a specific prefix, the server will identify the corresponding index-keys for this prefix by executing the prefix predicate function $\mathsf{SP^2E.PrefDec}$ over the encrypted index-keys, namely $addr$. Next, the clients decrypt the blocks (i.e. files) returned from the server to obtain the result. Note that the encryption (highlighted in Fig. 6) in our framework is designated for forward/backward privacy and can be varied according to different designs, such as PRF used in ref. [12,14], puncturable PRF in ref. [12,34,36] or encryption based on *oblivious data structure* in ref. [14,15,35]. While the framework allows different encryption, to enable prefix search, the message encrypted must include two parameters – seq and $\mathbf{UpdtCnt}[w]$), where $\mathbf{UpdtCnt}[w]$ stores the update operations on blocks in relation to an index-key w, and seq records the sequence of w in the index, hence making every block unique without compromising forward and backward privacy.

7 EXPERIMENTAL EVALUATION

In this section, we perform an in-depth analysis and report the performance of our scheme GridSE, comparing it with ST-ORAM [16], the state-of-the-art dynamic SSE for substring and plaintext search.

Setup. We implemented GridSE in Java using the JPBC [54] library for cryptographic operations, in particular, AES-256 as the PRF. All schemes were executed on a single machine while storing the database on RAM. We used an EC2 i3.xlarge AWS instance (Intel Xeon 2.30GHz CPU and 30.5GB RAM). The WAN experiment was run on two machines with a 30ms roundtrip time.

Experiment Overview. The experiment is conducted on the Gowalla location check-in dataset [55], a real-world dataset containing $6,442,890$ records contributed by $196,591$ worldwide users from which we picked $63,369$ distinct users within California by Google API. We tested for various database sizes $|DB| = 10^3$-10^7 by randomly choosing and duplicating users. The cell codes produced from different DGGS methods are evaluated such as Niemeyer's Geohash [2], Google S2 [3] and Uber H3 [4]. The letters of a cell code are from the textual alphabet. When building an index, we standardize the area of each cell to be $0.01km^2$ which is equivalent to a neighboring range of $100m$. Different DGGS methods yield various cell code lengths for a fixed cell area. Query ranges are tested spanning from $100m$ to $5km$ which associates the sizes from $100m^2$ to $100km^2$. We considered variable result between 10-10^5 blocks. Unless otherwise specified, after blocks were inserted, we delete at random 10% of the blocks matching the queried prefix, which is to emulate the impact of deletions on performance. All experimental values are the average obtained by 10 trials. To be precise, a location is kept with five digits such as $(\texttt{Long.} = -97.66712, \texttt{Lat.} = 30.20155)$. This unit of digits corresponds to $1m$ difference in reality.

Two parameters f and queried range are emphasized in our experiments. The queried range varies between $5km$ and

Table 1: Complexity of GridSE.

Computation Time		Storage		Communication Size	
Encryption	Search	Local State	Ciphertext	Update	Search
$O(N+d_w)$	$O(a_{w_p}+d_w)$	$O(d_w)$	$O(N+d_w)$	$O(1)$	$O(a_{w_p})$

d_w: the number of (distinct) index-keys in the encrypted index dictionary **InxDict** at server; N: the total number of updates; a_{w_p}: the number of updates in relation to the prefix w_p.

Table 2: Comparison evaluation for storage cost, search time and communication size.

Scheme	Local Storage	Search Time	Commu Size
Baseline (Plaintext)	-	$8.3ms$	$7.8KB$
GridSE	$30.7KB$	$20ms$	$15.3KB$
ST-ORAM (SOTA)	$424KB$	$5\times10^3 ms$	$10^4 KB$

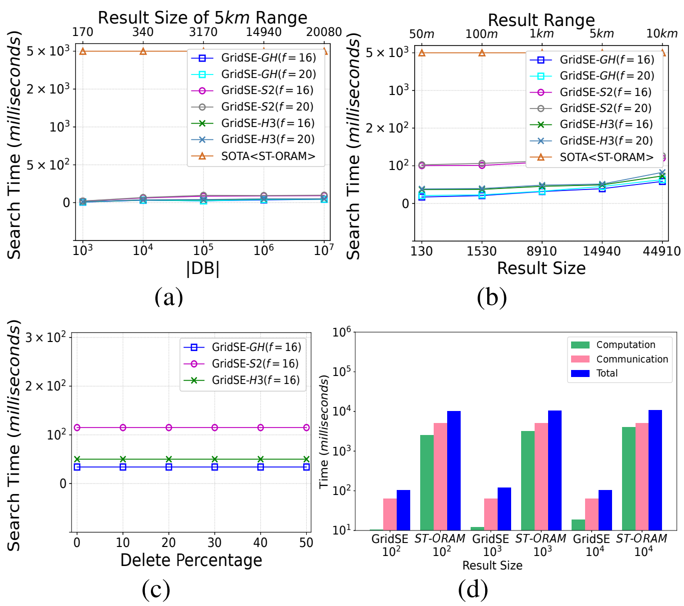


Figure 7: Computation time for (a) search vs. variable $|DB|$ sizes for result range $5km$, (b) search vs. variable result sizes, (c) search vs.% of deletions, (d) search under WAN with $40ms$ latency. (b)-(d) are all for $|DB| = 1M$.

$100m$ with the length of a queried prefix ranging from 5 to 10 characters. f is a parameter (in SP^{2}E) indicating a queried result valued of 16-20 bits *w.r.t* 256-bit AES ciphertexts.

7.1 Search Performance

We tested the computation time and communication size impacted from different f, queried ranges, deletions and WAN experiment for evaluating searches.

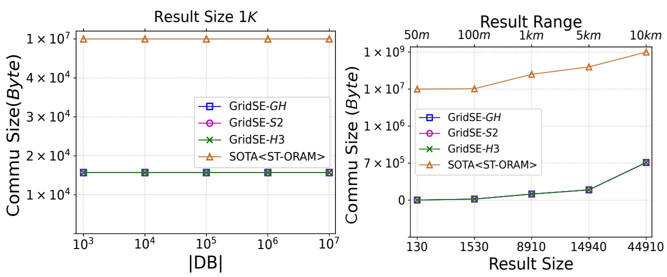
Computation Time. The computation time refers to the time needed for a single geographic search. As depicted in Fig. 7 (a)(b), GridSE and SOTA achieve nearly constant search time as the database size varies while the search time for both schemes slightly increases with the result size, as more blocks need to be returned and decrypted.

Comparison with the State of the Art. It is noteworthy that Fig. 7 (a) and (b) show a $150\times - 5000\times$ speedup of our scheme over ST-ORAM. This improvement is as expected, given the fact that ST-ORAM needs to scan every possible prefix within repeated tree structures, organized by every possible suffix (implemented with ORAM) which results in logarithmic cubical complexity. Instead, our solution leverages efficient symmetric encryption, maintaining almost constant complexity. Such improvement provides strong evidence for the feasibility and scalability of GridSE. For example, with a database of $1M$ blocks and a result size of 130, a geographic search over ciphertexts (including decryption) requires only $20ms$. Notably, the majority of the time is spent on the decryption while the server only needs to perform the lookups. This is proved by the unaffected performance with various parameter f as shown in Fig. 7, e.g., a 4-bit difference in f results in a negligible time difference of approximately $2ms$ for an encrypted database size of $|DB| = 1M$. Moreover, it is crucial to highlight the efficiency of GridSE, especially when achieving both forward and backward privacy.

Deletion Impact. One interesting observation is the independence of search time from the volume of deletions. As shown in Fig. 7 (c)., as the percentage of previous deletion increases from 0 to 50%, there is no notable variation in search time for a fixed database size of $1M$ and fixed result size of $10K$. This is because the deleted entries are not physically removed from the index. Notice that "deletion" is recorded in the parameter op stored within the index. Consequently, the deletion only influences the phase after decryption, i.e., to pick up the block identifiers that have been added but not yet deleted.

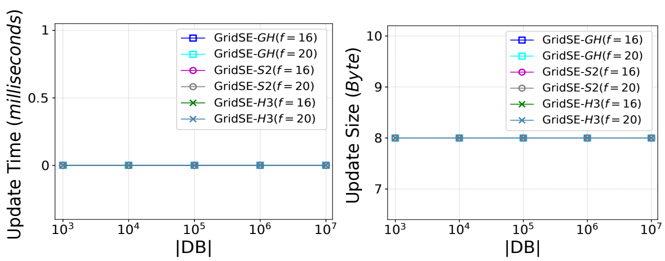
Experiment over WAN. To simulate the real-world scenario, we measure the end-to-end search time with separate server and client machines under the WAN setting. The latency between the server and the client is around 40ms. Fig. 7 (d) depicts the search time, breaking down to computation and communication, for database size $|DB| = 1M$. The data reported is for Niemeyer's Geohash. Although communication is the dominant overhead for both schemes, ST-ORAM imposes much heavier communication overhead due to the numerous interactive rounds with poly-logarithmic sizes transferred to the server while in our scheme, only 1 round of communication is needed. From our experiments, GridSE outperforms ST-ORAM by a factor of $67\times$ in terms of the result size.

Consider a database of size 10^6 with returned locations size of 10^3. Under such conditions, GridSE's performance metrics would be $83ms$ for communication, $18ms$ for computation, and a client storage requirement of $30.7KB$. This gives us a clearer grasp of the practical feasibility of our scheme.



(a) (b)

Figure 8: Communication cost for (a) search vs. variable $|DB|$ sizes for result range $5km$, (b) search vs. variable result sizes for $|DB| = 1M$.



(a) (b)

Figure 9: Update of (a) computation time, (b) communication size vs. variable $|DB|$ sizes.

Communication Cost. Fig. 8 shows the communication cost for a search under varying database sizes (a) and different result sizes (b). The communication cost grows linearly with the result size but is independent of the database size. Compared with ST-ORAM, GridSE saves 99% communication cost. This is achieved because in our scheme, any search contains only one round of communication between the client and server. The client only needs to send one "token" ($TList$) which is nearly constant in size and receive the result from the server. The communication cost can be further optimized by requiring the server to send the encrypted indexes back and then remove the deleted blocks, or asking the client to submit a cleanup process after each search.

7.2 Update Performance

Fig. 9 (a) and (b) show the update time and communication size for one update operation (add or deletion of a block) across varying database sizes, respectively. Notably, both the update time and communication size are constant ($5\mu s$ for update time and 8 bytes for update size) and independent of the database size. This is attributed to the constant communication cost incurred by GridSE, where only a key-value pair $(addr, val)$ needs to be transmitted from client to server, therefore highlighting the scalability of GridSE in real-world GIS systems.

7.3 Comparison with Plaintext Search

To further evaluate the efficiency of GridSE, we compare it with the plaintext prefix search for a database of size 10^6 and a result of 10^3. We implement this baseline via B-tree, the default index in MySQL for common keyword searches [56]. As shown in Table 2, GridSE incurs only $1.4\times$ additional computation cost and $0.9\times$ additional communication overhead compared to the plaintext search. This is because GridSE is built atop lightweight operations, e.g., XOR and hash functions. Considering its privacy-enhancing features, GridSE is promising in terms of real-time response.

8 Conclusion and Future Work

In this work, we propose GridSE, the first dynamic prefix symmetric searchable encryption (pSSE) scheme that supports fast secure geographic search with data dynamics and achieves backward and forward privacy simultaneously. We first introduce a new cryptographic primitive SP^{2}E for identifying whether a keyword contains a given prefix based on a lightweight cryptographic hash function. By leveraging the lightweight cryptographic operations, GridSE built atop SP^{2}E introduces almost constant overhead within a given geographical grid and is thus highly scalable. Experimental results show that GridSE achieves $150\times$ - $5000\times$ speedup as compared to the state of the art, and its performance is close to plaintext search, proving its feasibility for large-scale GIS/LBS systems.

Due to the constrained resources, we leave the implementation of GridSE in a real-world GIS/LBS system as a future work. We also provide several potential research directions here. While GridSE achieves the attractive prefix SSE for secure geographic search, which can also be used as generic prefix search primitive in other applications, a more versatile version - substring SSE is of great interest to the community but has not been supported yet. We point out that the lightweightness of GridSE is made possible by leveraging secure bit-wise operations. Combining such bit-level design with more complicated data structures, such as trees, might lead to a potential solution for substring SSE. In addition, extending GridSE to the multi-client setting is also a challenging but appealing direction.

References

[1] Wikipedia. Discrete global grid. https://en.wikipedia.org/wiki/Discrete_global_grid.

[2] Wikipedia. Geohash. http://en.wikipedia.org/wiki/Geohash/.

[3] Google s2. https://s2geometry.io/.

[4] Uber h3. https://www.uber.com/blog/h3/.

[5] Rui Wen, Yu Yu, Xiang Xie, and Yang Zhang. Leaf: A faster secure search algorithm via localization, extraction, and reconstruction. In ACM CCS 2020, pages 1219–1232.

[6] Seung Geol Choi, Dana Dachman-Soled, S. Dov Gordon, Linsheng Liu, and Arkady Yerukhimovich. Compressed oblivious encoding for homomorphically encrypted search. In ACM CCS 2021, pages 2277–2291.

[7] Ueli M. Maurer. Secure multi-party computation made simple. Discret. Appl. Math., 154(2):370–381, 2006.

[8] Simon Oya, Carmela Troncoso, and Fernando Pérez-González. Back to the drawing board: Revisiting the design of optimal location privacy-preserving mechanisms. In ACM CCS 2017, pages 1959–1972.

[9] Man Lung Yiu, Gabriel Ghinita, Christian S. Jensen, and Panos Kalnis. Enabling search services on outsourced private spatial data. VLDB J., 19(3):363–384, 2010.

[10] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In Applied Cryptography and Network Security, ACNS 2005, volume 3531, pages 442–455.

[11] Raphael Bost. Σοφος: Forward secure searchable encryption. In ACM CCS 2016, pages 1143–1154.

[12] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In ACM CCS 2017, pages 1465–1482.

[13] Kee Sung Kim, Minkyu Kim, Dongsoo Lee, Je Hong Park, and Woo-Hwan Kim. Forward secure dynamic searchable symmetric encryption with efficient updates. In ACM CCS 2017, pages 1449–1463.

[14] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In ACM CCS 2018, pages 1038–1055.

[15] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. Dynamic searchable encryption with small client storage. In NDSS 2020. The Internet Society.

[16] Tarik Moataz and Erik-Oliver Blass. Oblivious substring search with updates. IACR Cryptol. ePrint Arch., page 722, 2015.

[17] Boyang Wang, Ming Li, and Li Xiong. Fastgeo: Efficient geometric range queries on encrypted spatial data. IEEE Transactions on Dependable and Secure Computing, 16(2):245–258, 2019.

[18] Gabriel Ghinita and Razvan Rughinis. An efficient privacy-preserving system for monitoring mobile users: making searchable encryption practical. In ACM CODASPY 2014, pages 321–332.

[19] Kien Nguyen, Gabriel Ghinita, Muhammad Naveed, and Cyrus Shahabi. A privacy-preserving, accountable and spam-resilient geo-marketplace. In ACM SIGSPATIAL 2019, pages 299–308.

[20] Sina Shaham, Gabriel Ghinita, and Cyrus Shahabi. An efficient and secure location-based alert protocol using searchable encryption and huffman codes. In EDBT 2021, pages 265–276. OpenProceedings.org.

[21] Ruoyang Guo, Bo Qin, Yuncheng Wu, Ruixuan Liu, Hong Chen, and Cuiping Li. Mixgeo: efficient secure range queries on encrypted dense spatial data in the cloud. In ACM IWQoS 2019, pages 29:1–29:10.

[22] Ruoyang Guo, Bo Qin, Yuncheng Wu, Ruixuan Liu, Hong Chen, and Cuiping Li. Luxgeo: Efficient and security-enhanced geometric range queries. IEEE Trans. Knowl. Data Eng., 35(2):1775–1790, 2023.

[23] Yanguo Peng, Long Wang, Jiangtao Cui, Ximeng Liu, Hui Li, and Jianfeng Ma. LS-RQ: A lightweight and forward-secure range query on geographically encrypted data. IEEE Transactions on Dependable and Secure Computing, 19(1):388–401, 2022.

[24] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Theory of Cryptography, TCC 2007, volume 4392, pages 535–554. Springer.

[25] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In Theory of Cryptography, TCC 2009, volume 5444, pages 457–473. Springer.

[26] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In IEEE S&P 2000, pages 44–55.

[27] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In ESORICS 2015, volume 9327, pages 123–145. Springer.

[28] Florian Kerschbaum and Anselme Tueno. An efficiently searchable encrypted data structure for range queries. In ESORICS 2019, volume 11736, pages 344–364. Springer.

[29] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In ACM CCS 2006, pages 79–88.

[30] Ian Miers and Payman Mohassel. IO-DSSE: scaling dynamic searchable encryption to millions of indexes by improving locality. In NDSS 2017. The Internet Society.

[31] David Cash and Stefano Tessaro. The locality of searchable symmetric encryption. In EUROCRYPT 2014, volume 8441, pages 351–368. Springer.

[32] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In NDSS 2014. The Internet Society.

[33] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In USENIX Security 2016, pages 707–720. USENIX Association.

[34] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In ACM SIGSAC Conference on Computer and Communications Security, CCS'18, pages 763–780. ACM, 2018.

[35] Javad Ghareh Chamani, Dimitrios Papadopoulos, Mohammadamin Karbasforushan, and Ioannis Demertzis. Dynamic searchable encryption with optimal search in the presence of deletions. In USENIX Security 2022, pages 2425–2442. USENIX Association.

[36] Shi-Feng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K. Liu, Surya Nepal, and Dawu Gu. Practical non-interactive searchable encryption with forward and backward privacy. In NDSS 2021. The Internet Society.

[37] Thang Hoang, Attila A. Yavuz, and Jorge Guajardo. A secure searchable encryption framework for privacy-critical cloud storage services. IEEE Trans. Serv. Comput., 14(6):1675–1689, 2021.

[38] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In CRYPTO 2016, volume 9816, pages 563–592. Springer.

[39] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In Alfred V. Aho, editor, Annual ACM Symposium on Theory of Computing, 1987, pages 182–194.

[40] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. PHANTOM: practical oblivious computation in a secure processor. In ACM CCS 2013, pages 311–324.

[41] Syed Kamran Haider and Marten van Dijk. Flat ORAM: A simplified write-only oblivious RAM construction for secure processors. Cryptogr., 3(1):10, 2019.

[42] Thang Hoang, Rouzbeh Behnia, Yeongjin Jang, and Attila A. Yavuz. MOSE: practical multi-user oblivious storage via secure enclaves. In ACM CODASPY 2020, pages 17–28.

[43] Xiao Wang, T.-H. Hubert Chan, and Elaine Shi. Circuit ORAM: on tightness of the goldreich-ostrovsky lower bound. In ACM CCS 2015, pages 850–861.

[44] Vincent Bindschaedler, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, and Yan Huang. Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward. In ACM CCS 2015, pages 837–849.

[45] Thang Hoang, Muslum Ozgur Ozmen, Yeongjin Jang, and Attila A. Yavuz. Hardware-supported ORAM in effect: Practical oblivious search and update on very large dataset. Proc. Priv. Enhancing Technol., 2019(1):172–191, 2019.

[46] Zhiqiang Wu and Rui Li. OBI: a multi-path oblivious RAM for forward-and-backward-secure searchable encryption. In NDSS 2023. The Internet Society.

[47] Melissa Chase and Emily Shen. Substring-searchable symmetric encryption. Proc. Priv. Enhancing Technol., 2015(2):263–281, 2015.

[48] Iraklis Leontiadis and Ming Li. Storage efficient substring searchable symmetric encryption. In ACM AsiaCCS 2018, pages 3–13.

[49] Florian Hahn, Nicolas Loza, and Florian Kerschbaum. Practical and secure substring search. In ACM SIGMOD 2018, pages 163–176.

[50] Nicholas Mainardi, Davide Sampietro, Alessandro Barenghi, and Gerardo Pelosi. Efficient oblivious substring search via architectural support. In ACSAC 2020, pages 526–541. ACM.

[51] Tarik Moataz, Indrajit Ray, Indrakshi Ray, Abdullatif Shikfa, Frédéric Cuppens, and Nora Cuppens. Substring search over encrypted data. J. Comput. Secur., 26(1):1–30, 2018.

[52] Nicholas Mainardi, Alessandro Barenghi, and Gerardo Pelosi. Privacy preserving substring search protocol with polylogarithmic communication cost. In ACM ACSAC 2019, pages 297–312.

[53] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In ACM CCS 2012, pages 965–976.

[54] Jpbc libraries. http://gas.dia.unisa.it/projects/jpbc/#.Y8RjiuzMJpQ.

[55] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In ACM SIGKDD 2011, pages 1082–1090.

[56] MySQL. How mysql uses b-tree indexes. https://dev.mysql.com/doc/refman/8.0/en/index-btree-hash.html.

Appendix A Definition for SSE Forward/Backward Privacy

Forward Privacy. Forward privacy ensures that an update remains independent of past operations at the time this update occurs. Namely, previous searches will not leak any information about the blocks to be updated. A *forward private* scheme hides whether an addition is about a new block or one that might have been previously searched for.

***Definition* A.1** (*Forward Privacy of Dynamic SSE*). An $\mathcal{L}$-adaptively secure dynamic SSE scheme is forward private *iff* the leakage function $\mathcal{L}^{Updt}$ can be written as:

$$\mathcal{L}^{Updt}(op, w, id) = \mathcal{L}'^{Updt}(op, id) \tag{14}$$

where $\mathcal{L}'$ is a stateless function, w is an index key, op is insertion or deletion and id is a block identifier.

Backward Privacy. Backward privacy limits the information that the server can learn about blocks which are previously added but later deleted during searches related to them. In an ideal situation, the previously deleted blocks should not revealed to the adversary, at least the identifiers [14]. To capture backward privacy, we first follow the notation of [12] and then give the final definition.

Recall that an update is a tuple (u, op, in) where input $in = (w, id)$ with the modified block id, operation $op = add/del$, and u is the timestamp. For an index-key w, $\mathsf{TimeDB}(w)$ is a function that returns all timestamp/block-identifier pairs (of matching index-key w) that have been added to DB but not deleted later.

$$\begin{aligned}\mathsf{TimeDB}(w) = \{(u, id) | (u, add, (w, id)) \in Q \\ and\ \forall u', (u', del, (w, id)) \notin Q\},\end{aligned} \tag{15}$$

where Q is a list of queries that have been executed. An element of Q indicates a query of the form (u, w) where u is query timestamp, and w is the searched index-key.

The function $\mathsf{Updates(w)}$ returns the timestamp of all insertion and deletion operations on w in the previous queries Q. That is

$$\begin{aligned}\mathsf{Update}(w) = \{(u, id) | (u, add, (w, id)) \in Q \\ or\ (u, del, (w, id)) \notin Q\}.\end{aligned} \tag{16}$$

$\mathsf{DelHist(w)}$ is a function of capturing previously deleted entries, which returns the timestamp of all insertion and deletion operations. The output of it reveals which deletion corresponds to which addition.

$$\begin{aligned}\mathsf{DelList}(w) = \{(u^{add}, u^{del}) | (u, add, (w, id)) \in Q \\ or\ (u, del, (w, id)) \notin Q\}.\end{aligned} \tag{17}$$

From the above functions, we give the formal definition of backward privacy with the three types of leakage, from Type-I which reveals information the least to Type-III which leaks the most. A parameter a_w is introduced here, which indicates the total number of updates associated with w.

***Definition* A.2** (*Backward Privacy of Dynamic SSE*). An $\mathcal{L}$-adaptively secure dynamic SSE scheme is backward private:

$$\begin{aligned}
&\textbf{BP-Type-I}: \textit{iff}\ \mathcal{L}^{Updt}(op, w, id) = \mathcal{L}'(op),\ and \\
&\qquad \mathcal{L}^{Srch}(w) = \mathcal{L}''(\textbf{TimeDB}(w), a_w). \\
&\textbf{BP-Type-II}: \textit{iff}\ \mathcal{L}^{Updt}(op, w, id) = \mathcal{L}'(op, w),\ and \\
&\qquad \mathcal{L}^{Srch}(w) = \mathcal{L}''(\textbf{TimeDB}(w), \textbf{Updates}(w)). \\
&\textbf{BP-Type-III}: \textit{iff}\ \mathcal{L}^{Updt}(op, w, id) = \mathcal{L}'(op, w),\ and \\
&\qquad \mathcal{L}^{Srch}(w) = \mathcal{L}''(\textbf{TimeDB}(w), \textbf{DelHist}(w)).
\end{aligned}$$

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless functions.

Note that the above definition assumes the leakage of actually retrieving blocks will be allowed, namely the blocks that currently contain an index-key w. Specifically, the function **TimeDB** reveals the identifiers of the blocks matching w.

Appendix B Security Experiments of SP^{2}E

Fig. 10 presents the security experiments for encryption and token generation of SP^{2}E in respective.

Appendix C Correctness of $\mathsf{PrefDec}$ if $w_p \sqsubseteq w$

Fig. 11 shows the output formula of $\mathsf{PrefDec}$, that correctly exposes the bits 0^f if $w_p \sqsubseteq w$, for which we highlight these bits using the color blue. The correctness of $w_p \sqsubseteq w$ is hold since there exists an indicator if the current keyword w satisfies the prefix query requirement, and 0^f is such an indicator.

Appendix D Correctness of $\mathsf{PrefDec}$ if $w_p \not\sqsubseteq w$

If $w_p \not\sqsubseteq w$, the correctness of $\mathsf{PrefDec}$ should guarantee that the output of $\mathsf{PrefDec}$ is an indistinguishable value without any *indicator-bits*, namely consecutive bits valued at zero. We consider two cases under this condition of $w_p \not\sqsubseteq w$, one is the first letters of w_p are the prefix of w, but w_p is not a prefix of w, e.g., $w_p =$ "$apps$", $w =$ "$apple$"; and the other is any composed parts of the sequence in w_p is not a prefix of w, e.g., $w_p =$ "$star$", $w =$ "$apple$". To formulate both cases, we introduce a set J for denoting each sequence of the character of the difference set w_J, that is $\{w - w_p | w_p\}$ in w_p. For instance, $w_J =$'s' and $J = \{4\}$ if $w_p =$ "$apps$" and $w =$ "$apple$", because 's' is the different letter in w_p compared to w; and 4 indicates the sequence of 's' in w_p. We have $1 \leq |J| \leq |w_p|$, and $|J|$ is the size of J. The output of $\mathsf{PrefDec}$ if $w_p \not\sqsubseteq w$ can be derived to the formula in Fig. 12. This formula shows that there exists no *indicator-bits* exposed in the decrypted result. Thus, the correctness of $\mathsf{PrefDec}$ for $w_p \not\sqsubseteq w$ is validated.

$\underline{\mathbf{Expt}^{\mathsf{IND-CPA}}_{\mathcal{A},\mathsf{SP^2E.Enc}}(\lambda):}$

$(SK, MSK) \leftarrow \mathsf{KeyGen}(1^\lambda, t)$; table $U, V \leftarrow \emptyset$
$(w_0, w_1, st) \leftarrow \mathcal{A}'^{O^{\mathsf{Enc}}_{SK,seq}(\cdot,\cdot)}(1^\lambda)$, *s.t.* $|w_0|, |w_1| \le t$
$b \xleftarrow{\$} \{0,1\}$;
$\hat{ct} \leftarrow \mathsf{Enc}(SK, \mathsf{PreEnc}(MSK, seq, w_b))$
$b' \leftarrow \mathcal{A}^{O^{\mathsf{Enc}}_{SK,seq}(\cdot,\cdot)}(st, \hat{ct})$
Return $(b' = b)$.

$\underline{O^{\mathsf{Enc}}_{SK,seq}(w):}$

$\mathbf{m} \leftarrow \mathsf{PreEnc}(MSK, seq, w)$; $ct, \delta \leftarrow \mathsf{Enc}(SK, \mathbf{m})$
$U \leftarrow U \cup \mathbf{m}$,
$V \leftarrow V \cup (\delta, seq)$
Return ct.

$\underline{\mathbf{Expt}^{\mathsf{IND-}f\mathsf{-CPA}}_{\mathcal{A},\mathsf{SP^2E.TKGen}}(\lambda):}$

$(SK, MSK) \leftarrow \mathsf{Setup}(1^\lambda, t)$; table $W \leftarrow \emptyset$
$(w_{p_0}, w_{p_1}, st) \leftarrow \mathcal{A}'^{O^{\mathsf{TKGen}}_{SK,MSK,\delta,seq}(\cdot,\cdot)}(1^\lambda)$, *s.t.* $|w_{p_0}|, |w_{p_1}| \le t$, $|w_{p_0}| = |w_{p_1}|$
$b \xleftarrow{\$} \{0,1\}$;
$\hat{k}' \leftarrow \mathsf{TKGen}(SK, MSK, \delta, seq, w_{p_b})$
$b' \leftarrow \mathcal{A}^{O^{\mathsf{TKGen}}_{SK,MSK,\delta,seq}(\cdot,\cdot)}(st, \hat{k}')$
Return $(b' = b)$.

$\underline{O^{\mathsf{TKGen}}_{SK,MSK,\delta,id}(w_p):}$

$k' \leftarrow \mathsf{TKGen}(SK, MSK, \delta, seq, w_p)$
$W \leftarrow W \cup (\delta, seq)$
Return k'.

Figure 10: Semantic Security of Encryption and Token Generation in SP^{2}E

$$
\begin{aligned}
k' \oplus ct = 1:\ &= \bigoplus_{i=1}^{|w_p|} \Big(0^{(i-1)f} \parallel H(sk_{w_{p_i}} \parallel seq \parallel i)\Big).\mathsf{sub}(0, l(\lambda) \oplus \Big(\eta.\mathsf{sub}(0, pos_1) \parallel \delta.(pos_1, pos_2) \parallel \eta.\mathsf{sub}(pos_2, l(\lambda))\Big) \\
&\oplus \bigoplus_{i=1}^{|w|} \Big(0^{(i-1)f} \parallel H(sk_{w_i} \parallel seq \parallel i)\Big).\mathsf{sub}(0, l(\lambda)) \oplus \Big(\delta.\mathsf{sub}(0, pos_1) \parallel \delta.(pos_1, pos_2) \parallel \delta.\mathsf{sub}(pos_2, l(\lambda))\Big) \\
&\mathit{Note\ that:}\ \text{ we have } sk_{w_{p_i}} = sk_{w_i}\ \mathit{for}\ 1 \le i \le |w_p|,\ \mathit{if}\ w_p \sqsubseteq w, \\
2:\ &= \bigoplus_{i=|w_p|+1}^{|w|} \Big(0^{(i-1)f} \parallel H(sk_{w_i} \parallel seq \parallel i)\Big).\mathsf{sub}(0, l(\lambda)) \oplus \Big((\eta \oplus \delta).\mathsf{sub}(0, pos_1) \parallel 0^f \parallel (\eta \oplus \delta).\mathsf{sub}(pos_2, l(\lambda))\Big) \\
&\mathit{Note\ that:} \text{ for simplicity, we denote the part before } '\oplus' \text{ as } D_H = \bigoplus_{i=|w_p|+1}^{|w|} \Big(0^{(i-1)f} \parallel H(sk_{w_i} \parallel seq \parallel i)\Big), \\
3:\ &= (\eta \oplus \delta).\mathsf{sub}(0, pos_1) \parallel 0^f \parallel \Big(D_H \oplus (\eta \oplus \delta)\Big).\mathsf{sub}(pos_2, l(\lambda))
\end{aligned} \tag{18}
$$

Figure 11: Expanding the output $k' \oplus ct$ of PrefDec if $w_p \sqsubseteq w$

$$
\begin{aligned}
k' \oplus ct\ = 1:&= \text{Step 1 is as the same as the step 1 in Figure 11.} \\
2:&= \bigoplus_{i}^{J} \Big(0^{(i-1)f} \parallel \big(H(sk_{w_{p_i}} \parallel seq \parallel i) \oplus H(sk_{w_i} \parallel seq \parallel i)\big)\Big).\mathsf{sub}(0, l(\lambda)) \oplus \\
&\bigoplus_{i=|w_p|+1}^{|w|} \Big(0^{(i-1)f} \parallel H(sk_{w_i} \parallel seq \parallel i)\Big).\mathsf{sub}(0, l(\lambda)) \oplus \Big((\eta \oplus \delta).\mathsf{sub}(0, pos_1) \parallel 0^f \parallel (\eta \oplus \delta).\mathsf{sub}(pos_2, l(\lambda))\Big), \\
&\mathit{Note\ that:} \text{ for simplicity, we denote } S_H = \bigoplus_{i}^{J} \Big(0^{(i-1)f} \parallel \big(H(sk_{w_{p_i}} \parallel seq \parallel i) \oplus H(sk_{w_i} \parallel seq \parallel i)\big)\Big), \\
&D_H = \bigoplus_{i=|w_p|+1}^{|w|} \Big(0^{(i-1)f} \parallel H(sk_{w_i} \parallel seq \parallel i)\Big) \text{ and } J \text{ is the sequence of } w_J, \\
3:&= \Big(S_H \oplus (\eta \oplus \delta)\Big).\mathsf{sub}(0, pos_1) \parallel S_H.\mathsf{sub}(pos_1, pos_2) \parallel \Big((S_H \oplus D_H) \oplus (\eta \oplus \delta)\Big).\mathsf{sub}(pos_2, l(\lambda)).
\end{aligned} \tag{19}
$$

Figure 12: Expanding the output $k' \oplus ct$ of PrefDec if $w_p \not\sqsubseteq w$

Appendix E Security Analysis of SP^{2}E: Proof for Thm 5.5

In the following, Game$_i$ denotes the i-th game.

Game_0: this is exactly the real security game of token security (Def. 5.3) between the challenger and a PPT adversary $\mathcal{A}$. More concretely, the challenger $\mathcal{C}$ executes the setup algorithm and generates $SK = (sk_1, sk_2), MSK = (sk_{c_1}, \cdots, sk_{c_{|\Lambda|}} ..., f)$ where sk_{c_i} is the secret key associated with a character and $|\Lambda|$ is the total number of characters in alphabet Λ. Next, The challenger runs the pre-encryption and encryption algorithm to generate the corresponding local parameter δ for each keyword w.

The challenger then responds to the token queries for prefix w_p from $\mathcal{A}$. For each token query, the adversary submits the queried prefix w_p for which the challenger computes the corresponding δ and returns the token k' by running $\mathsf{TKGen}(SK, MSK, \delta, seq, w_p)$. Then, the adversary submits two prefix w_{p_0} and w_{p_1} to challenger who flips a coin at random to select b from $\{0,1\}$. The challenger generates a token $\hat{k}'$ for prefix w_{p_b} and sends it to $\mathcal{A}$. The adversary $\mathcal{A}$ can further make a polynomial number of times queries to the token generation oracle for any w_p except w_{p_0} and w_{p_1}. The adversary finally submits a guess b'. By the Def. 5.3, if a cryptographic hash function is modeled as a random oracle, we have that

$$Adv^{\text{IND-}f\text{-CPA}}_{\mathcal{A},\mathrm{SP^2E.TKGen}}(\lambda) = \left| Pr_{G_0}[b' = b] - \frac{1}{2} \right|. \tag{20}$$

Game_1: the difference of this game with Game_0 is that each sub-value (except for the zero-prefix) of $\mathbf{m}$ is chosen uniformly at random, rather than computed from calling the hash function H. In this game, each sub-value is computed as

(1) $r_1, r_2, ...r_i \xleftarrow{\$} \mathcal{K}$, where $i = 1, \cdots, |w|$.

(2) $sub_i = 0^{(i-1)f} \parallel r_i$. Specifically, $sub_1 = r_1$.

The challenger then compute $\mathbf{m}^* = \sum_{i=1}^{|w|}(sub_i)$ and answer the queries with $\mathbf{m}^*$ instead of $\mathbf{m}$.

LEMMA E.1. Game_1 *and* Game_0 *are computationally indistinguishable in the random oracle model. That is*

$$\left| Pr_{G_1}[b' = b] - Pr_{G_0}[b' = b] \right| \le Adv^{PRF}_{\mathcal{A}',F}(\lambda) \tag{21}$$

Game_2: it is the same as Game_1, except that $\mathbf{m}^*$ is replaced by the first sub-value sampled uniformly at random, not the sum of randomly sampled sub-values. That is

(1) $sub_1 = r_1 \xleftarrow{\$} \mathcal{K}$,

The challenger then answers the token queries with $\mathbf{m}^* = sub_1$. This game is obviously identical to the previous one under the random oracle model. Thus, it holds that

$$Pr_{G_2}[b' = b] = Pr_{G_1}[b' = b]. \tag{22}$$

Game_3: this game differs from Game_2 when generating the challenge token $\hat{k}'$ given $\mathbf{m}$ and corresponding δ. To simplify the notation, we re-write the token $\hat{k}'$ in Game_0 into two strings by applying the bounded-f representation. That is, $\hat{k}' = \{\hat{k}'_1, \hat{k}'_2\}$, where $\hat{k}'_1 = (\mathbf{m} \oplus \delta).\mathsf{sub}(pos_1, pos_2)$ is a f-bit string, and $\hat{k}'_2 = (\mathbf{m} \oplus G(sk_2, seq)).\mathsf{sub}(0, pos_1) \parallel (\mathbf{m} \oplus G(sk_2, seq)).\mathsf{sub}(pos_2, l(\lambda))$ is the remaining part of $\hat{k}'$.

In this game, the challenger first samples k_2 uniformly at random from $\mathcal{K}$ instead of using $G(sk_2, \mathbf{m})$. Similarly, the challenger samples $k_1 = H_2(sk_1, \mathbf{m}^*)$ where $H_2 : \mathcal{K} \times \mathbb{N}* \to \mathcal{K}$ is a hash function. The challenger then answers the token query and prepares the challenge token in the following ways:

- For a token query for prefix w_p, when $\mathbf{m} = \mathbf{m}^*$, the challenge outputs a token $\hat{k}' = \{\hat{k}'_1, \hat{k}'_2\}$ where $\hat{k}'_1 = (\mathbf{m}^* \oplus k_1).\mathsf{sub}(pos_1, pos_2)$ and $\hat{k}'_2 = (\mathbf{m}^* \oplus k_2).\mathsf{sub}(0, pos_1) \parallel (\mathbf{m}^* \oplus k_2).\mathsf{sub}(pos_2, l(\lambda))$. Otherwise, the challenger outputs $\hat{k}' = \mathsf{TKGen}(SK, MSK, G(sk_1, \mathbf{m}^*), seq, w_p)$.

- To generate the challenge token, the challenger first random selects a bit b from $\{0,1\}$. The challenger computes the token by invoking $\mathsf{TKGen}(SK, MSK, G(sk_1, \mathbf{m}^*), seq, w_{p_b})$

LEMMA E.2. *If F is a secure PRF and H is a one-way cryptographic hash function, then* Game_3 *and* Game_2 *are computationally indistinguishable. That is*

$$\left| Pr_{G_3}[b' = b] - Pr_{G_2}[b' = b] \right| \le Adv^{PRF}_{\mathcal{A},F}(l(\lambda) - f) + Adv^{H}_{\mathcal{A},H}(f) \tag{23}$$

Note that the advantage of $\mathcal{A}$ on Game_3 is negligible under the IND-CPA security of the underlying symmetric encryption $\mathsf{SE}\text{-}f$ on f-bit bounded, for which we formally claim it as

LEMMA E.3. *If SE-f is an IND-CPA secure symmetric encryption scheme, then it holds that*

$$\left|Pr_{G_3}[b'=b]-\frac{1}{2}\right| = Adv^{\text{IND-CPA}}_{\mathcal{A},\mathsf{SE}\text{-}f}(\lambda) \tag{24}$$

Assuming that all lemmata hold, then we have

$$\begin{aligned}
&Adv^{\text{IND-}f\text{-CPA}}_{\mathcal{A},\text{SP}^2\text{E}}(\lambda)\\
&= \mathsf{MAX}(Adv^{\text{IND-CPA}}_{\mathcal{A},\text{SP}^2\text{E.Enc}}(\lambda), Adv^{\text{IND-}f\text{-CPA}}_{\mathcal{A},\text{SP}^2\text{E.TKGen}}(\lambda))\\
&= Adv^{\text{IND-}f\text{-CPA}}_{\mathcal{A},\text{SP}^2\text{E.TKGen}}(\lambda)\\
&= |Pr_{G_0}[b'=b]-\frac{1}{2}| \qquad (25)\\
&\le |Pr_{G_1}[b'=b]-Pr_{G_0}[b'=b]|\\
&+|Pr_{G_2}[b'=b]-Pr_{G_3}[b'=b]|+|Pr_{G_3}[b'=b]-\frac{1}{2}|\\
&\le Adv^{PRF}_{\mathcal{A},F}(l(\lambda)-f)+Adv^{H}_{\mathcal{A},H}(f)+Adv^{\text{IND-CPA}}_{\mathcal{A},\mathsf{SE}\text{-}f}(\lambda).
\end{aligned}$$

To hold the above deduction, we are required to validate all lemmata. Lemma E.1 is straightforward to be proved by the reduction from H and PRF to random oracle and validated by the security of random oracle, showing that Game_0, Game_1 and Game_2 are computationally indistinguishable. The reduction from Game_2 to Game_3 is validated through Lemma E.2, for which the only difference from Game_2 to Game_3 is the replacement for k'. As we stated in the above proof, the component of k' not bounded with f-bit is replaced by random strings while that of k' bounded with f-bit is by calling H functions. By combining all the proofs, we can get that the IND-f-CPA security of SP^{2}E holds in the random oracle model (cf. Theorem 5.5). In addition, though f-bit bounded, our scheme achieves the requirement of adaptive security as it does not require any limits on an adversary issuing queries in the challenge phase.

Appendix F Security Analysis of GridSE: Proof for Thm 6.1

As described in Appendix A, the backward privacy of dynamic SSE can be defined with different types of leakage. We focus on Type-II backward privacy of GridSE which is formally stated in Thm 6.1. The proof is similar to that of Thm 3.1 in ref. [14]. The main difference is that the security of GridSE is reduced to the adaptive security of SP^{2}E which is stated in Thm 5.5. We finish the proof by constructing a simulator via a sequence of games and show that the advantage of a PPT adversary against our protocol is negligible. The behavior of our simulator Sim is described here. For setup. Sim simply follows the setup algorithm defined in GridSE. During an update query, the simulator generates the transcript $(addr, val)$ by sampling uniformly at random in the range of corresponding PRF G. The simulator stores the entry $I(j) = (addr, val, in)$ where j is the timestamp of the update. If the index j does not correspond to a valid update, Sim sets $I(j)$ to be null. During a search, Sim receives the leakage function $\mathsf{TimeDB}(w_p)$ and $\mathsf{Update}(w_p)$. The simulator infers from $\mathsf{Update}(w_p)$ the timestamps of previous updates related to the searched prefixes and sends the search results to the server. Upon receiving, the simulator calls $\mathsf{TimeDB}(w_p)$ to retrieve the sets of documents that currently contain the searched prefixes and sends them to the server. We now prove the security of GridSE as follows:

Game_0: This is exactly the same as the real SSE security game described in Def 4.1. Thus, we have

$$Pr[\text{REAL}^{\Sigma}_{\mathcal{A}}(\lambda,q)=1] = Pr[\mathsf{Game}_0=1]$$

Game_1: This is exactly the same as Game_0 except $G_K(I, \mathbf{BCnt}[w])$ which is computed by a PRF G and key K. It is replaced by a value sampled uniformly at random from the range of G. The indistinguishability between Game_1 and Game_0 is guaranteed by the security of PRF. That is,

$$|Pr[\mathsf{Game}_1=1]-Pr[\mathsf{Game}_0=1]| \le Adv^{\mathsf{PRF}}_{\mathcal{A},G}(\lambda)$$

Game_2: This is exactly the same as Game_1 except the value of $addr$. Instead of calling the pre-encryption and encryption algorithms of SP^{2}E, the value of $addr$ is substituted by a uniformly random sampled value in $\mathcal{Y}$. A list I with q entries is maintained with each entry $I(j) = (addr, val, in)$ for $j = 1, \cdots, q$ indicating a valid update operation storing the random values sampled together with the operation input $in = (w, id)$. For all j that does not correspond to an update operation, the entries $I(j)$ would be null. When performing a search for prefix w_p, the game scans the list to identify the entries that match the prefix restriction that $w_p \sqsubseteq w$ and sends the corresponding $addr$ to the server to receive results. It then scans I again to deduce R_{w_p} the set of documents that currently holds index-keys w with prefix w_p and sends R_{w_p} to the server. The indistinguishability of Game_2 and Game_1 is guaranteed by the adaptive security of SP^{2}E which has already been proved in Appendix E. Hence, we have

$$|Pr[\mathsf{Game}_2=1]-Pr[\mathsf{Game}_1=1]| \le d\cdot Adv^{\text{IND-}f\text{-CPA}}_{\mathcal{A},\text{SP}^2\text{E}}(\lambda)$$

where d indicates the maximum length of a keyword.

Game_3: This is exactly the same as IDEAL^{PSSE} with simulator described above. The transcripts produced in this game follow the same distribution as those generated in Game_2 since the leakage functions correspond to the same values that would be computed in Game_2 and the value of val is distributed uniformly at random. Thus, we have

$$\begin{aligned}Pr[\text{REAL}^{\Sigma}_{\mathcal{A}}(\lambda,q)=1]-Pr[\text{IDEAL}^{\Sigma}_{\mathcal{A},\mathsf{Sim}}(\lambda,q)] \le Adv^{\mathsf{PRF}}_{\mathcal{A},G}(\lambda)\\ +d\cdot Adv^{\text{IND-}f\text{-CPA}}_{\mathcal{A},\text{SP}^2\text{E}}(\lambda)\end{aligned}$$

which completes the proof.