PathGES: An Efficient and Secure Graph Encryption Scheme for Shortest Path Queries

PathGES: An Efficient and Secure Graph Encryption Scheme for Shortest Path Queries[*]

Francesca Falzon
ETH Zürich
Zürich, Switzerland
ffalzon@ethz.ch

Esha Ghosh
Microsoft Research
Redmond, WA, USA
esha.ghosh@microsoft.com

Kenneth G. Paterson
ETH Zürich
Zürich, Switzerland
kenny.paterson@inf.ethz.ch

Roberto Tamassia
Brown University
Providence, RI, USA
roberto@tamassia.net

ABSTRACT

The increasing importance of graph databases and cloud storage services prompts the study of private queries on graphs. We propose PathGES, a graph encryption scheme (GES) for single-pair shortest path queries. PathGES is efficient and mitigates the state-of-the-art attack by Falzon and Paterson (2022) on the GES by Ghosh, Kamara, and Tamassia (2021), while only incurring an additional logarithmic factor in storage overhead. PathGES leverages a novel data structure that minimizes leakage and server computation.

We generalize what it means for one leakage function to leak less than another by defining a relation with respect to a family of query sequences and show that our scheme provably leaks less than the GKT scheme when all queries have been issued. We complement our security proof with a cryptanalysis that demonstrates an information-theoretic gap in the size of the query reconstruction space of our scheme as compared to the GKT scheme and provide concrete examples of the gap for several graph families. Our prototype implementation of PathGES is efficient in practice for real-world social network and geographic data sets. In comparison with the GKT scheme, PathGES has on average the same response size and up to 1.5x faster round-trip query time.

KEYWORDS

Graph Databases; Encrypted Databases; Searchable Encryption

1 INTRODUCTION

Graphs model data in numerous large-scale real-world applications ranging from fraud detection to biological networks to recommendation systems. Plaintext graph databases have consequently been heavily studied and widely deployed in industry including Facebook Tao [64], Amazon Neptune [1], GraphDB [55], and Neo4j [53].

When a graph database is outsourced to a cloud service, one naive solution for privacy is to upload the encrypted database and download it each time a query is issued. An alternative solution is to use strong, but computationally expensive cryptographic techniques such as fully-homomorphic encryption [24] or oblivious RAM [29]. These approaches are either bandwidth inefficient or not yet scalable, which begs the question of whether efficient server-side query processing on an encrypted graph is possible.

Chase and Kamara introduced **structured encryption (STE)** [14] as a generalization of **searchable symmetric encryption (SSE)** [3, 6, 9, 10, 14, 17, 25, 33, 34, 52, 63]. STE enables the encryption of structured data such that the data can be privately queried in sublinear time. STE gains its efficiency by sacrificing security to an extent: these schemes leak some information about the queries or underlying data. Security of STE is typically proven using the real-ideal paradigm, which states that the scheme does not leak any information beyond some well-defined leakage function. A long line of work has shown that such leakage can be detrimental to the security of schemes supporting key-word search (e.g., [8, 18, 56]) and range queries (e.g.[21, 30, 35, 38–40, 49]), emphasizing the importance of cryptanalyzing proposed schemes.

A **graph encryption scheme (GES)** is a form of STE that enables one to encrypt a graph, outsource it to an untrusted server, and then process queries over the encrypted graph. These schemes are efficient, often incurring only a constant overhead over plaintext variants. Our understanding of GESs is still in its infancy, and existing schemes typically support only a single query type, such as adjacency queries [14], approximate shortest distance queries [50], or single-pair shortest path queries [26].

A **single-pair shortest path** (**SPSP**) query on a graph G takes as input a pair of vertices, u and v, of G and outputs a shortest path between u and v. SPSP queries have important applications to social network analysis, routing, resource management, and biology [67]. To support SPSP queries, one could use the trivial solution of running the all-pairs-shortest-paths algorithm on G, computing a dictionary that maps each query to the corresponding shortest path, and then using a standard encrypted multimap scheme (such as [9]) to encrypt the dictionary. Although this approach is straightforward to implement and offers optimal bandwidth complexity, the storage overhead is quadratic. As a result, alternative GES constructions have been proposed to reduce this storage overhead, while still offering similar security guarantees. One such work is the GES by Ghosh, Kamara, and Tamassia [26] – hence forth referred to as the **GKT scheme** – which supports SPSP queries.

While there are a variety of leakage abuse attacks against STE schemes supporting range queries, work on attacks against GESs is limited. Assuming a passive server-side honest-but-curious adversary that knows the plaintext graph, G, and observes SPSP queries issued on the GKT scheme for G, Falzon and Paterson [23] recently demonstrated an attack that aims at reconstructing the plaintext queries issued by the client. Their attack is optimal in the absence of auxiliary information. It recovers the plaintext queries up to an equivalence set of queries that are indistinguishable from the leakage. They show that this set can be as small as 3 to 5 queries on real-world social network graphs.

We propose PathGES, a new GES supporting SPSP queries. Our scheme has quantifiably less leakage for a large class of graphs than the GKT scheme and is designed in light of the attack of Falzon and Paterson [23]. The key to our reduced leakage is our new data structure for SPSP queries that minimizes leakage and server computation. First, we compute the spanning shortest-path

[*] This is the full version of work accepted to CCS 2024.

Scheme	Scheme Complexity			Attack	Size of Query Reconstruction Space			
	Resp size	Query time	Space	Citation	K_n	L_n	S_n	G_n
GKT [26]	t	t	n^2	[23]	$(n!)^{n+1}$	$2^{n/2}$	1	$\leq \left(2^{\frac{7n-2\sqrt{n}+4}{4}}\right)$
PathGES	$2t$	$\log n + 2t$	$n^2 \log n$	–	$(n^2)!$	$\geq (n!)^{\log n - 1}$	$\geq n \prod_{i=0}^{\log(n+1)-3} (2^i!)$	$\geq \left(\frac{\sqrt{n}}{4}!\right)^{\frac{\sqrt{n}}{2}n}$

Table 1: Comparison of our PathGES **scheme with the GKT scheme [26] for SPSP queries. Here, n is the number of vertices of the graph, t is the length of the shortest path returned by a query, K_n is the complete graph, L_n is the line graph and S_n is the "asymmetric star" with one central vertex and $\log(n+1) - 1$ incident paths of lengths $2^1, \ldots, 2^{\log(n+1)-1}$, and G_n is the grid graph of size $\sqrt{n} \times \sqrt{n}$.**

trees rooted at each node in the graph G that we wish to encrypt. These trees are decomposed into edge-disjoint paths, which are further processed into a set of fragments and stored in a sequence of two encrypted multimaps.

Unlike in the GKT scheme, PathGES prevents a server-side adversary from computing the query trees (trees that are isomorphic to spanning trees of the underlying graph G and whose paths correspond to the queried shortest paths) in all but trivial cases, and thus defends against the first essential step of the attack in [23]. Informally, the **query reconstruction space** of a GES is the number of unique assignments of plaintext queries to each issued query that produces that same leakage. We characterize the query reconstruction space of our scheme and show that for many graph families, the gap in the size of the query reconstruction space between our scheme and the GKT scheme is at least super-polynomial in the number of vertices in G (See Table 1). We further explain why query recovery against our scheme reduces to the graph isomorphism problem, a problem which is not known to be polynomial-time solvable for general graphs.

We generalize the order relation on leakage functions by Bost and Fouque [5] to families of query sequences; our relation implies an upper bound on the size of the query reconstruction space. We then show that our scheme leaks less than the GKT scheme for the family of query sequences in which each SPSP query has been issued at least once. While this is a slightly weaker notion, we note that many leakage functions are not directly comparable under Bost and Fouque's definition since the order relation on leakage functions is a partial ordering [5]. This highlights the subtleties of comparing two complex leakage functions.

Our scheme offers $O(n^2 \log n)$ storage overhead (vs. $O(n^3)$ for the trivial solution). Compared to the GKT scheme, it offers a reduced leakage function at the expense of only a logarithmic factor increase in storage and slightly longer setup times (i.e., at most 4.1× constant factor increase for the real-world datasets tested). PathGES also achieves optimal asymptotic bandwidth complexity, incurring no more than a 2x constant factor over the trivial solution.

We support our scheme with a proof of security, cryptanalysis, and an implementation to demonstrate the scheme's practicality on real-world datasets. On average, the response size of PathGES is similar to that of the GKT scheme, while its query time is up to 1.5x faster for longer paths with the potential for further speed-up through parallelism. In contrast, the GKT scheme's query algorithm cannot be parallelized due to its inherently sequential processing.

1.1 Prior work

Schemes. Meng et al. [50] present three schemes that leverage sketch-based distance oracles to support shortest distance queries. Liu et al. [47] and Shen et al. [61] also use distance oracles to precompute the shortest distances between queries; the former uses order-preserving encryption for efficiency at the expense of leaking the orders of the distances at setup, and the latter only supports constrained shortest path distance queries. Wang et al. [65] utilize additive homomorphic encryption and garbled circuits to encrypt graph data and support shortest distance queries, however, their scheme relies on a third party, the proxy. Ghosh et al. [26] describe a scheme based on the SP-matrix of the graph and whose setup time, query time, and storage are asymptotically optimal. An attack on this scheme was presented in [23].

Most recently, Chamani et al. [12] propose GraphOS; in contrast to PathGES which preprocesses the shortest paths in a multimap and then encrypts the data structure, GraphOS takes a more general approach that uses a trusted server-side enclave and doubly-oblivious primitives to access the vertices/edges of a graph and execute graph queries in an online manner. Their scheme assumes a different threat model and leaks less information than our scheme, at the expense of multiple expensive oblivious look-ups. As an example, querying for the shortest path using GraphOS requires running Dijkstra's algorithm on the encrypted graph and the query time is thus orders of magnitude larger than the standard multimap lookup of our scheme (e.g., $10^{2.5}$-10^3s for a graph with 2^{13} nodes, compared to $10^{2.9}$-$10^{3.4}\mu$s for a graph of $2^{13.4}$ nodes for our scheme, when run on broadly comparable hardware).

A number of STE schemes supporting other query types and graphs have been described such as GES supporting top-k-nearest neighbors [46], STE for conceptual graphs [57], STE for knowledge graphs [44], and dynamic STE for bipartite graphs [41, 45]. Solutions for graph queries in other security models have also been proposed, but are outside the scope of this work. These approaches include SGX [20], private information retrieval [66], differential privacy [60], and structural anonymization [7].

Attacks. Goetschmann [27] presents the first leakage abuse attack on a GES, which succeeds against one of the schemes by Meng et al. [50] by estimating distances between nodes of the graph and then using this information to infer the issued queries. Falzon and Paterson [23] give a query recovery attack against the GKT scheme [26]. They show that a server-side adversary with knowledge of the plaintext graph G and the leakage of certain queries can recover the set of possible plaintext queries for each issued query.

1.2 Contributions

In this work, we let $G = (V, E)$ denote a graph, $n = |V|$ the number of vertices, and $m = |E|$ the number of edges. An SPSP query on a graph can be answered online using a single-source shortest path algorithm, such as Dijkstra's algorithm which runs in time $O(m + n \log n)$. This approach is very costly when multiple queries are processed. To avoid such overhead, one can pre-process the graph and store the shortest path information in a data structure such as an SP-matrix [16] or distance oracle [15]. Existing attacks are against GESs that use the SP-matrix or distance oracles. One could instead use other pre-processing techniques for answering SPSP queries e.g. [28]. However, a number of nearby nodes must still be scanned; this could result in more leakage or require the use of other cryptographic primitives and we leave this as future work.

We propose a new data structure that mitigates previous attacks. This data structure enables our scheme to be non-interactive whilst minimizing server computation, resulting in fast query time in theory and practice. Let $G = (V, E)$ be the graph that we wish to encrypt. At a high level, our GES works as follows. For each vertex $v \in V$ we compute the shortest path tree rooted at v whose paths correspond to shortest paths in G; we denote this tree as T_v. We then decompose each tree T_v into edge-disjoint paths.

We could stop here and store the disjoint paths, however, the path returned might still have length $O(n)$. Given a disjoint path p, we extract subpaths of p of lengths that are powers of two, called canonical fragments. These fragments are stored in a multimap. A second multimap associates SPSP queries with the identifiers of fragments used to assemble the shortest path for the given query. The multimaps are encrypted using standard encrypted multimap schemes. The fragments ensure that the bandwidth is asymptotically optimal and the indirection keeps storage costs down by only storing each fragment once.

To query the graph, the server computes one lookup to learn the identifiers of the fragments and a second lookup to retrieve the actual fragments. This is done non-interactively in one roundtrip between the client and server by using a response-revealing encrypted multi-map (EMM) to encrypt the first multimap. Our approach is computationally and bandwidth efficient, and mitigates the attack from prior work [23].

Our contributions can be summarized as follows:

- We present PathGES, a non-interactive GES for SPSP queries with reduced query leakage. (Section 4)
- We describe a new data structure for responding to SPSP queries. This data structure contributes to the enhanced security of PathGES whilst maintaining optimal bandwidth complexity. (Section 4)
- We introduce a generalized relation on leakage functions with respect to a family of query sequences and prove that our scheme leaks less than the GKT scheme for the family of query sequences in which each SPSP query is issued at least once. (Section 5)
- We support our scheme with a thorough cryptanalysis and prove that for several graph families, the size of the reconstruction space of PathGES is strictly greater (by a superpolynomial factor) than that of GKT when all queries have been issued. (Section 5)
- We evaluate the performance of PathGES on a number of real-world datasets and show improved round-trip query time over the GKT scheme. (Section 6)

2 PRELIMINARIES

Notation. For an integer n, let $[n] = \{1, 2, \ldots, n\}$. We denote the concatenation of strings a and b as $a||b$. Given a set X, we use $x \leftarrow\$ X$ to denote that the element x was sampled uniformly at random from X. We denote the security parameter using λ and the state using st. We denote a function negligible in λ by $\mathsf{negl}(\lambda)$. Unless otherwise stated, we write $\log n$ for $\log_2 n$.

Graphs. A **graph** $G = (V, E)$ comprises of a set of vertices V (of size n) and a set of edges $E = V \times V$ (of size m). If the graph is **simple** then the pairs of vertices in E are unordered and if the graph is **directed**, then the pairs of vertices in E are ordered. A **tree** $T = (V, E)$ is a graph that is connected and contains no cycles. A **rooted tree** $T = (V, E, r)$ is a tree in which a special ("labeled") node $r \in V$ is singled out; we refer to r as the root of the tree.

A **single pair shortest path (SPSP) queries** on a graph $G = (V, E)$. An SPSP query is the evaluation of a function SPSP that takes as input two vertices $u, v \in V$, and outputs a path $p_{u,v} = (u, w_1, \ldots, w_t, v)$ of minimal length in G if u and v are connected, and outputs $\perp$ otherwise. For simplicity, we assume connected graphs, however, our scheme applies directly to general graphs.

Dictionaries and Multimaps. A **dictionary** D is a map from a label space $\mathbb{L}$ to a value space $\mathbb{V}$. A **multimap** is a generalization of a dictionary in which each label may be associated with multiple values. Formally, a multimap M is a map from a label space $\mathbb{L}$ to the powerset of a value space $2^{\mathbb{V}}$. If lab $\mapsto$ val then we write val $\leftarrow$ D[lab]. We denote the assignment of val to lab as D[lab] $\leftarrow$ val (and correspondingly for multimaps).

2.1 Graph Encryption Scheme

Definition 1. *A **graph encryption scheme** is a tuple of algorithms* GES = (KeyGen, Encrypt, Token, Search, Reveal) *with the following syntax:*

- KeyGen *is probabilistic; it takes a security parameter λ and outputs a secret key K.*
- Encrypt *is probabilistic; it takes a secret key K and a graph G and outputs an encrypted database* **ED**.
- Token *takes a key K and query q and returns a search token* tk.
- Search *takes an encrypted database* **ED** *and a search token* tk *and outputs a response* resp.
- Reveal *takes a key K and response* resp *and outputs plaintext m.*

This is a purely syntactical definition. In practice, algorithms KeyGen, Encrypt, Token, and Reveal are executed by the client, and Search is executed by the server.

A graph encryption scheme for SPSP queries GES is correct if for all graphs $G = (V, E)$ and all SPSP queries $q = (u, v) \in V \times V$, if $K \leftarrow \mathsf{GES.KeyGen}(1^\lambda)$ and $\mathbf{ED} \leftarrow \mathsf{GES.Encrypt}(K, G)$, then $m \leftarrow \mathsf{GES.Reveal}(K, \mathsf{resp})$ is a shortest path from u to v in G where $\mathsf{tk} \leftarrow \mathsf{GES.Token}(K, q)$, $\mathsf{resp} \leftarrow \mathsf{GES.Search}(\mathbf{ED}, \mathsf{tk})$.

2.1.1 Security. Security of GESs is parameterized by a leakage function $\mathcal{L} = (\mathcal{L}_S, \mathcal{L}_Q)$ that specifies an upper bound on the information leaked at setup ($\mathcal{L}_S$) and at query time ($\mathcal{L}_Q$). We define security using the real-ideal paradigm with respect to passive persistent adversaries who execute the protocol honestly and who learn the output of the leakage functions. Such adversaries include an

```
Real$^{\mathsf{GES}}_{\mathcal{A}}(1^\lambda)$

Initalize($1^\lambda$)
  $G \leftarrow \mathcal{A}$
  $K \leftarrow \mathsf{GES.KeyGen}(1^\lambda)$
  $\mathbf{ED} \leftarrow \mathsf{GES.Encrypt}(K, G)$
  return ED
Query($q$)
  $\mathsf{tk} \leftarrow \mathsf{GES.Token}(K, q)$
  return tk
Finalize($b$)
  return $b$
```

```
Ideal$^{\mathsf{GES}}_{\mathcal{A},\mathcal{S}}(1^\lambda)$

Initalize($1^\lambda$)
  $G \leftarrow \mathcal{A}$
  $(\alpha, \mathsf{st}_{\mathcal{L}}) \leftarrow \mathcal{L}_{\mathsf{S}}(G)$
  $(\mathbf{ED}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}(\alpha)$
  return ED
Query($q$)
  $(\alpha, \mathsf{st}_{\mathcal{L}}) \leftarrow \mathcal{L}_{\mathsf{Q}}(G, q)$
  $(\mathsf{tk}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}(\alpha)$
  return tk
Finalize($b$)
  return $b$
```

Figure 1: Games $\mathbf{Real}^{\mathsf{GES}}_{\mathcal{A}}$ and $\mathbf{Ideal}^{\mathsf{GES}}_{\mathcal{A},\mathcal{S}}$.

honest-but-curious server or a network adversary that has compromised the communication channel. For the purpose of cryptanalysis and to be comparable to previous attack works [23, 27], we are concerned with query privacy, though we emphasize that the goal of a GES is also to hide the graph. We assume that the adversary knows or even chooses the graph G and we wish to prevent the adversary from inferring the plaintext queries from the leakage. The public graph assumption is standard for schemes that support private graph queries [26, 51, 60]. This setting is ideal for routing scenarios in which the road network may be public (e.g., Google Maps), but the routing information of users is sensitive.

Definition 2. *Let* GES *be a graph encryption scheme and let* $\mathcal{L} = (\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Q}})$ *be a tuple of stateful algorithms. We say that* GES *is* ***$\mathcal{L}$-secure*** *if for all polynomial-time adversaries* $\mathcal{A}$*, there exists a polynomial-time simulator* $\mathcal{S}$ *such that*

$$|\Pr[\mathbf{Real}^{\mathsf{GES}}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Ideal}^{\mathsf{GES}}_{\mathcal{A},\mathcal{S}}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda).$$

and games $\mathbf{Real}^{\mathsf{GES}}_{\mathcal{A}}(\lambda)$ *and* $\mathbf{Ideal}^{\mathsf{GES}}_{\mathcal{A},\mathcal{S}}(\lambda)$ *are defined as in Figure 1.*

2.2 Encrypted Multimap Scheme

Encrypted multimap schemes are a fundamental building block of our GES. They allow one to encrypt and outsource a multimap, and then later query labels in the multimap. EMM schemes can be either response-revealing or response-hiding. Informally, a response-revealing scheme reveals the plaintext answer of a query to the server whereas a response-hiding scheme does not.

Definition 3. *A* ***response-hiding encrypted multimap (EMM) scheme*** *is a tuple of algorithms* EMM-RH = (KeyGen*,* Encrypt*,* Token*,* Get*,* Reveal) *with the following syntax:*

- KeyGen *is probabilistic and takes a security parameter* λ*, and outputs a secret key* K*.*
- Encrypt *is probabilistic and takes a key* K *and multimap* M*, and outputs encrypted multimap* **EM***.*
- Token *takes a key* K *and a label* lab*, and outputs a search token* tk*.*
- Get *takes a search token* tk *and an encrypted multimap* **EM** *and returns response* resp*.*
- Reveal *takes a key* K *and response* resp *and returns a set of plaintext values* $\{\mathsf{val}_i\}_{i \in [k]}$*.*

A **response-revealing EMM scheme** EMM-RR comprises of four algorithms (KeyGen, Encrypt, Token, Get) such that KeyGen, Encrypt, and Token are as in Definition 3 and Get instead takes as input an encrypted multimap **EM** and a search token tk and returns a set of plaintext values $\{\mathsf{val}_i\}_{i \in [k]}$. Our GES makes black-box use of one response-revealing and one response-hiding EMM scheme. We assume the storage complexity of the EMMs is linear in the number of values in the plaintext multimap. We ignore the number of bits needed to encode the actual values themselves as this is typically smaller than the security parameter (which we can take to be a large constant in practice).

Correctness requires that for all multimaps M and labels lab in M, if $K \leftarrow \mathsf{EMM.KeyGen}(\lambda)$, $\mathbf{EM} \leftarrow \mathsf{EMM.Encrypt}(K, \mathsf{M})$, then executing $\mathsf{EMM.Reveal}(K, \mathsf{resp})$ such that $\mathsf{tk} \leftarrow \mathsf{EMM.Token}(K, \mathsf{lab})$ and $\mathsf{resp} \leftarrow \mathsf{EMM.Get}(\mathbf{EM}, \mathsf{tk})$, results in the output $\mathsf{M}[\mathsf{lab}]$.

2.2.1 *Security.*

The security of an EMM scheme is also defined using the real-ideal paradigm. The $\mathbf{Real}^{\mathsf{EMM}}_{\mathcal{A}}$ and $\mathbf{Ideal}^{\mathsf{EMM}}_{\mathcal{A}}$ games are the same as in Figure 1, except that the adversary picks M, Initialize is executed using EMM.KeyGen and EMM.Encrypt, and Query is executed using EMM.Token.

2.2.2 *Leakage Functions of EMMs.*

We consider EMM schemes with the following standard leakage.

- **Multimap size** (Size): This setup leakage refers to the total number of values in the multi-map. Formally, for a multimap M with label space $\mathbb{L}$, $\mathsf{Size}(\mathsf{M}) = \sum_{\mathsf{lab} \in \mathbb{L}} |\mathsf{M}[\mathsf{lab}]|$.
- **Query pattern** (QP): This reveals whether two queries are equal. Let M be a multimap and $\mathsf{lab}_1, \ldots, \mathsf{lab}_k$ be a sequence of queries. Then $\mathsf{QP}(\mathsf{M}, (\mathsf{lab}_1, \ldots, \mathsf{lab}_k)) = A$ where A is a $k \times k$ matrix of bits such that $A[i, j] = 1$ if and only if $\mathsf{lab}_i = \mathsf{lab}_j$.
- **Access pattern** (AP): This reveals the individual values returned for each query. Let M be a multimap and $\mathsf{lab}_1, \ldots, \mathsf{lab}_k$ be a sequence of queries. Then $\mathsf{AP}(\mathsf{M}, (\mathsf{lab}_1, \ldots, \mathsf{lab}_k)) = (\mathsf{M}[\mathsf{lab}_i])_{i \in [k]}$.
- **Volume pattern** (Vol): Reveals the number of values returned per query. Let M be a multimap and $\mathsf{lab}_1, \ldots, \mathsf{lab}_k$ be a sequence of queries. Then $\mathsf{Vol}(\mathsf{M}, (\mathsf{lab}_1, \ldots, \mathsf{lab}_k)) = (|\mathsf{M}[\mathsf{lab}_i]|)_{i \in [k]}$.

Our GES requires two EMM schemes, EMM-RR and EMM-RH; EMM-RR is response-revealing to make the GES non-interactive and EMM-RH is response-hiding for added security. For concreteness, we assume EMM-RR to be (Size, (QP, AP))-secure and EMM-RH to be to be (Size, (QP, Vol))-secure.

3 TECHNICAL BACKGROUND

The leakage function of the GKT scheme leaks the edges along which two paths with the same destination vertex intersect. This leakage is detrimental to the security of the queries and Falzon and Paterson [23] demonstrate how the plaintext queries can be recovered from this leakage. Our goal is to reduce this leakage.

Let $G = (V, E)$ be the graph that we wish to encrypt. We first compute n shortest path trees $\{T_v\}_{v \in V}$ e.g. using an **all-pairs shortest paths (APSP)** algorithm: each tree T_v is rooted at a vertex $v \in V$ and the paths correspond to shortest paths in G. These trees are then decomposed into edge-disjoint paths. Our scheme can be instantiated using *any* algorithm that decomposes trees into edge-disjoint paths. To concretize our schemes we employ a data-structure technique called **heavy-light decomposition (HLD)** [62] to decompose the trees (we describe the algorithm in detail in Appendix A). HLD is a general and standard method for decomposing trees into disjoint paths and answering graph queries. HLD is well-studied, easily implementable, and offers good efficiency guarantees, thus making it a good candidate for deployment.

We start by proving that many common APSP algorithms output collections of paths from which spanning rooted shortest path trees can be computed. This is a necessary condition, since we must apply HLD to a rooted tree.

Definition 4. *Let $G = (V, E)$ be a graph and P be a collection of paths of G. The* ***graph induced by*** *P is defined as $G|_P = (V_P, E_P)$ where $V_P = \{v : v \in e, e \in p, p \in P\}$ and $E_P = \{e : e \in p, p \in P\}$. By definition, $V_P \subseteq V$ and $E_P \subseteq E$.*

Lemma 5. *Let $G = (V, E)$ be a graph and P be the collection of paths that results from running Floyd-Warshall on G.* [1] *Let P_v denote the set of paths in P that terminate at v. Then for all $v \in V$, the graph $G|_{P_v}$ forms a tree with v as its root.*

The proof can be found in Appendix B.1.

Corollary 6. *Let $G = (V, E)$ be a graph and P the collection of paths resulting from running Floyd-Warshall on G. For any vertex $v \in V$, if two paths $p, p' \in P_v$ coincide at a vertex u, then they must coincide at each vertex along the path from u to v.*

We denote tree $G|_{P_v}$ as T_v and refer to it as the **single destination shortest path (SDSP) tree** for v in G.

After computing n SDSP trees, one for each v in V, our scheme decomposes each tree into edge-disjoint paths using HLD. At a high level, HLD works by labeling edges in a tree T as either "heavy" or "light" such that the light edges demarcate where an edge-disjoint path ends and a new path starts. HLD guarantees that any path in T crosses no more than $\log n$ disjoint paths. Applying HLD to a tree may, in the worst case, result in an edge-disjoint path of length $O(n)$. This happens when most edges are heavy e.g. when G is a path. To address this, our scheme computes and stores what we call the *canonical fragments* of each edge-disjoint path. The canonical fragments of a path $p_{u,v}$ are the subpaths ending at v whose lengths are powers of two. This allows us to optimize bandwidth at the expense of a small additional overhead in storage.

Definition 7. *Let $p_{u,v}$ be a path padded up to the next smallest power of 2 such that the padding vertices are pre-pended to u. Let $p_{u,v}^{(j)}$ denote the subpath comprised of the last 2^j edges in $p_{u,v}$. The* ***canonical fragments*** *of $p_{u,v}$ are $\{p_{u,v}^{(j)} : 0 \leq j \leq \log_2 |p_{u,v}|\}$.*

The dummy vertices should be clearly demarcated from the real ones so that they can be filtered out at decryption time. How this is done would depend on the low-level representation used for the vertices. For example, one could label the vertices of G with integers in $[n]$ and the dummy vertices with 0.

4 PATHGES: A GES FOR SPSP QUERIES

We now present PathGES, a GES for SPSP queries that leverages indirection to ensure non-interactivity and reduce leakage. The pseudocode can be found in Figure 2.

4.1 Scheme Description

4.1.1 High-level Description. Given a graph $G = (V, E)$, we compute the set of SDSP trees $\{T_v\}_{v \in V}$ and decompose each tree into disjoint paths (e.g., using HLD). We then encode the shortest path information as a set of fragments which can be retrieved using a sequence of look-ups to two multimaps. The first multimap M_1 maps each query to a set of search tokens derived from the fragment identifiers used to look up the corresponding canonical fragments in the second multimap; this map is encrypted using a response-revealing EMM scheme. The second multimap M_2 maps each fragment identifier to the respective canonical fragment; it is encrypted using a response-hiding EMM scheme. We denote the encrypted versions of M_1 and M_2 as $\mathbf{EM}_1$ and $\mathbf{EM}_2$, respectively.

To issue query $(u, v) \in V \times V$, the client computes a query-specific search token $\mathsf{tk}_{u,v}$ which the server uses to retrieve the token set T from $\mathbf{EM}_1$. For each token $\mathsf{tk} \in T$, the server retrieves the corresponding encrypted fragment from $\mathbf{EM}_2$ and adds it to the response. By property of HLD, each shortest path is comprised of no more than $\log n$ fragments and, thus, search requires at most $\log n$ look-ups to $\mathbf{EM}_2$ (one look-up for each search token obtained from $\mathbf{EM}_1$) to retrieve at most $2t$ edge where t is the length of the shortest path. Note that the 2x factor results from padding the fragments up to the next power of 2. The server returns the set of encrypted fragments to the client who then decrypts them to recover the shortest path.

4.1.2 Formal description. KeyGen takes as input a security parameter λ and returns a pair of keys (K_1, K_2), one each for $\mathbf{EM}_1$ and $\mathbf{EM}_2$, respectively.

Encrypt takes as input key $K = (K_1, K_2)$ and the graph $G = (V, E)$ that the client wishes to encrypt. It initializes empty multimaps M_1 and M_2. For each vertex $r \in V$ it computes the SDSP tree rooted at r to obtain T_r. For each T_r, the client decomposes the tree into edge-disjoint paths e.g. using the HLD algorithm (Algorithm 8). We denote the decomposed tree as T_r^D.

Importantly, the disjoint paths are processed in a breadth first search (BFS) manner, so that the multimap can be computed in one traversal of each tree. For each path $p_{u,v}$ in T_r^D, as it is discovered, and for each canonical fragment $p_{u,v}^{(j)}$ of $p_{u,v}$, the client:

1. Generates token $\mathsf{tk} \leftarrow \mathsf{EMM\text{-}RH.Token}(K_2, (r, u, v, j))$ using the response-hiding EMM;
2. Sets $\mathsf{M}_2[(r, u, v, j)] \leftarrow p_{u,v}^{(j)}$;
3. For each non-pad vertex w in $p_{u,v}^{(j)} \setminus p_{u,v}^{(j-1)}$, sets $\mathsf{M}_1[(w, r)] \leftarrow \mathsf{M}_1[(u, r)] \cup \{\mathsf{tk}\}$.

The key is that the fragments needed to reconstruct the shortest path from u to r comprises the fragment from u to v, and the fragments of the shortest path from v to r. Since T_r^D is processed in a BFS manner, $\mathsf{M}_1[(r, u)]$ has already been computed. Each label in M_2 is associated with a set of values describing a canonical fragment; each value consists of an edge in the fragment. How the values are encrypted depends on the EMM scheme used.

Next, the multimaps are padded to prevent leaking information about the underlying graph at setup. See Section 4.2 for a discussion. M_1 is padded up to $n^2 \lfloor \log n \rfloor$ entries where each entry

[1] We prove this for the Floyd-Warshall algorithm, but this property holds for other algorithms like those of Bellman-Ford and Dijkstra [16].

```
// Generate secret key.
KeyGen($1^\lambda$) → $K$
    $K_1$ ← EMM-RR.KeyGen($1^\lambda$)
    $K_2$ ← EMM-RH.KeyGen($1^\lambda$)
    return $(K_1, K_2)$

// Compute the encrypted database.
Encrypt($K, G$) → ED
    Initialize multimaps $M_1$ and $M_2$
    Parse $(K_1, K_2) \leftarrow K$
    for $r \in V$ do
        Compute SDSP tree $T_r$ rooted at $r$ in $G$
        $T_r^D$ ← ComputeHld($T_r, r$)
        for each subpath $p_{u,v} \in T_r^D$ in BFS manner do
            Let $\ell$ be the next power of 2 greater than $|p_{u,v}|$
            Pad $p_{u,v}$ to length $\ell$
            for $j \in [0, \lceil \log_2 \ell \rceil]$ do
                // Compute fragment and add it to $M_2$.
                Let $p_{u,v}^{(j)}$ comprise the last $2^j$ edges of $p_{u,v}$
                $M_2[(r, u, v, j)] \leftarrow p_{u,v}^{(j)}$
                if $j = 0$ then
                    $s \leftarrow p_{u,v}^{(0)}$
                else $s \leftarrow p_{u,v}^{(j)} \setminus p_{u,v}^{(j-1)}$
                for non-pad vertex $w$ in $s$ do
                    // Add tokenset of query $(w, r)$ to $M_1$
                    tk ← EMM-RH.Token($K_2, (r, u, v, j)$)
                    $M_1[(w, r)] \leftarrow M_1[(v, r)] \cup \{$tk$\}$
                    Permute $M_1[(w, r)]$
    Pad $M_1$ and $M_2$ to $n^2 \log n$ and $4n^2$, respectively.
    $\mathbf{EM}_1$ ← EMM-RR.Encrypt($K_1, M_1$)
    $\mathbf{EM}_2$ ← EMM-RH.Encrypt($K_2, M_2$)
    return $(\mathbf{EM}_1, \mathbf{EM}_2)$

// Compute search token.
Token_C($K, (u, v)$) → tk
    Parse $(K_1, K_2) \leftarrow K$
    tk ← EMM-RR.Token($K_1, (u, v)$)
    return tk

// Look up shortest path.
Search_S(ED, tk) → resp
    Initialize resp ←⊥
    Parse $(\mathbf{EM}_1, \mathbf{EM}_2)$ ← ED
    // Retrieve respective search tokens from $\mathbf{EM}_1$.
    $T$ ← EMM-RR.Get($\mathbf{EM}_1$, tk)
    // Retrieve respective fragments from $\mathbf{EM}_2$.
    for tk′ $\in T$ do
        $c$ ← EMM-RH.Get($\mathbf{EM}_2$, tk′)
        resp ← resp $\cup \{c\}$
    return resp

// Recover the shortest path.
Reveal_C($K$, resp) → $p$
    Parse $(K_1, K_2) \leftarrow K$
    Initialize $P \leftarrow \emptyset$
    // Decrypt each fragment.
    for $c \in$ resp do
        $m$ ← EMM-RH.Reveal($K_2, c$)
        Unpad $m$
        $P \leftarrow P \cup \{m\}$
    // Process the collection of fragments.
    Sort $P$ into path $p$ from $u$ to $v$
    return $p$
```

Figure 2: Psuedocode for our scheme, PathGES**, which supports single pair shortest path queries over an encrypted graph.**

comprises of λ bits. One could implement this by adding sufficiently many dummy pairs to M_1 such that each pair consists of a single dummy value, e.g. the all-zero bit string of length λ. In M_2, each fragment is stored as a set of edges; these edges form the set of values to be encrypted by EMM-RH. M_2 is thus padded by adding sufficiently many dummy single-edge fragments until the total length of the fragments (real and dummy) in the multimap is $4n^2$. If the vertices in G are represented as integers in $[n]$, then the edges of the dummy fragments can be encoded as the pair $(0, 0)$ (recall that EMM-RH uses randomized encryption). The maps are encrypted to obtain $\mathbf{EM}_1$ ← EMM-RR.Encrypt(K_1, M_1) and $\mathbf{EM}_2$ ← EMM-RH.Encrypt(K_2, M_2).

To query for $(u, v) \in V \times V$, the client computes a search token $\mathsf{tk}_{u,v}$ ← EMM-RR.Token($K_1, (u, v)$) which it sends to the server. The server uses $\mathsf{tk}_{u,v}$ to look up the token set T in $\mathbf{EM}_1$ that is needed to retrieve the corresponding encrypted canonical fragments in $\mathbf{EM}_2$. The server initializes empty set resp and for each tk′ $\in T$ adds the encrypted fragment c ← EMM-RH.Get($\mathbf{EM}_2$, tk′) to resp. Finally, resp is returned to the client who can then decrypt and sort the fragments to recover the shortest path.

4.1.3 *A remark on the use of Indirection.*

"Multimap chaining" was first introduced by Chase and Kamara [14] and has been used in a number of works to support various queries [11, 31, 54]. Using the layered multimap approach to enable complex queries and reduce storage costs is a standard technique in the structured encryption literature. The technique uses at least two multimaps to index data structures such that the tokens for accessing one multimap are stored as values in another. Without this approach, our scheme would require storing a copy of each fragment for each query that the fragment corresponds to. Moreover, since we are able to encrypt the first multimap using a response revealing EMM scheme, we are able to make query processing completely non-interactive.

4.2 Complexity and Correctness

The run time of Encrypt is upper bounded by the time needed to compute the SDSP trees, which for general graphs takes time $O(n^3)$. Token entails a single call to the EMM-RR.Token. Our chosen implementation of EMM-RH only requires two PRF evaluations [9]. Assuming use of HLD to decompose the trees, Search takes time $O(\log n + t)$, since any shortest path results in at most $1 + \log n$

look-ups to the EMMs to retrieve at most $2t$ edges. In the worst case, the server must return $2t$ edges, where t is the length of the shortest path and so Reveal takes time $O(t)$.

Encrypted multimaps $\mathbf{EM}_1$ and $\mathbf{EM}_2$ use $O(n^2 \log n)$ and $O(n^2)$ space, respectively, for a total storage complexity of $O(n^2 \log n)$. We pad the multimaps up to the worst case size to prevent additional leakage. In particular, M_1 must be padded up to $n^2\lfloor \log n \rfloor$ values and M_2 must be padded up to $4n^2$ values. The upper bounds for the sizes of M_1 and M_2 are met by the binary tree and the cycle graph, respectively, and thus our bounds are tight. The proofs of these bounds can be found in Appendix B.2 and B.3.

Theorem 8. *Let $G = (V, E)$ be a graph on n vertices, λ be the security parameter, and* **ED** *be the result of encrypting G using* PathGES *(Figure 2). Executing* Encrypt *on G takes $O(n^3)$ time and produces an encrypted database of size $O(n^2 \log n)$. Generating a token* tk *for a query $(u, v) \in V \times V$ using* Token *requires $O(1)$ time, where* tk *is of size $O(\lambda)$. Executing* Search *on* tk *and* **ED** *takes time $O(\log n + t)$ and produces a response* resp *of at most 2t encrypted edges, where* resp *decrypts to the shortest path from u to v in G and where t is the length of the queried shortest path.* Reveal *runs in $O(t)$ time.*

The proof can be found in Appendix B.4.

4.3 Leakage

The leakage of PathGES is a function of the leakage of the underlying EMMs. For concreteness, we assume EMM-RR to be $(\mathsf{Size}, (\mathsf{QP}, \mathsf{AP}))$-secure and EMM-RH to be to be $(\mathsf{Size}, (\mathsf{QP}, \mathsf{Vol}))$-secure. Recall that the EMM schemes' Encrypt algorithms are probabilistic by definition and repeated values are not leaked at setup.

Setup leakage. At setup, the scheme leaks the size of the EMMs, which is a function of n. We thus have that

$$\boxed{\mathcal{L}_\mathsf{S}(G) = n.}$$

Query leakage. Given a sequence of k SPSP queries, PathGES leaks whether two queries are equal, whether two (encrypted) fragments in a response are equal, and the length of each fragment. The **query pattern** (QP) of a sequence of SPSP queries $q_1, \ldots, q_k$ is a matrix $A \in \{0, 1\}^{k \times k}$ such that $A[i, j] = 1$ if and only if q_i and q_j have the same source and destination vertex.

Our scheme also leaks which canonical fragments co-occur in the response of each query. In their work on range search schemes over encrypted multi-attribute data, Falzon et al. [22] identified a form of leakage called *structure pattern*, i.e. the co-occurrence of subqueries used to respond to each range query. The co-occurrence of subranges in the schemes of [22] is analogous to the co-occurrence of fragments in our scheme, and thus a similar idea applies here.

Structure pattern can be viewed as a function of the underlying EMM's leakage and the underlying search data-structure. Formally, the **structure pattern** (Str) of a sequence of SPSP queries $q_1, \ldots, q_k$ can be viewed as a weighted bipartite graph $H = (I \cup F, E')$ where I is the set of possible queries, F is the set of fragment identifiers,

$$E' = \left\{ (q, f) \in I \times F \ : \ \begin{array}{l} \text{Encrypted fragment } f \text{ is in} \\ \text{the response of query } q. \end{array} \right\}$$

and for each $(q, f) \in E'$, $weight((q, f))$ is the length of f. As queries are issued, the adversary can update H online.

In total, the query leakage of PathGES (Figure 2) is

$$\boxed{\mathcal{L}_\mathsf{Q}(G, ((u_1, v_1), \ldots, (u_k, v_k))) = (\mathsf{QP}, \mathsf{Str}).}$$

Discussion on leakage. Similar to previous schemes, the setup leaks the number of vertices in the graph. The query leakage of our scheme, however, is more nuanced. It leaks the *set* of encrypted canonical fragments. Since M_2 is encrypted using a response-hiding EMM scheme that leaks query and volume pattern, the encrypted fragments only leak the search pattern (as a result of retrieving each fragment) and the length of the fragment. Consider queries $q = (u, v)$ and $q' = (u', v')$ associated with shortest paths p and p'. If the responses to q and q' have one or more fragments in common, then the following conditions hold:

(1) The destination nodes v and v' are equal.
(2) For some edge-disjoint path p'' in T_v and integer ℓ, p and p' intersect p'' in $\geq \lceil \ell/2 \rceil$ and $\leq \ell$ edges.

Note that condition (2) implies the underlying shortest paths intersect in *at least* one edge.

In Figure 3d we depict the (encrypted) fragments produced when encrypting the graph in Figure 3a with PathGES and querying all queries with destination vertex 1. The path of length 3 is padded up to length 4 (padding nodes are denoted with a dotted border). The fragments' search tokens are permuted before being adding to M_1.

4.4 Security

Theorem 9. *Let* EMM-RR *and* EMM-RH *be* $(\mathsf{Size}, (\mathsf{QP}, \mathsf{AP}))$*-secure and* $(\mathsf{Size}, (\mathsf{QP}, \mathsf{Vol}))$*-secure encrypted multimap schemes, respectively. If* PathGES *(Figure 2) is instantiated using* EMM-RR *and* EMM-RH*, then it is* $(\mathcal{L}_\mathsf{S}, \mathcal{L}_\mathsf{Q})$*-secure according to Definition 2 for* $(\mathcal{L}_\mathsf{S}, \mathcal{L}_\mathsf{Q}) = (n, (\mathsf{QP}, \mathsf{Str}))$.

The proof can be found in Appendix B.5

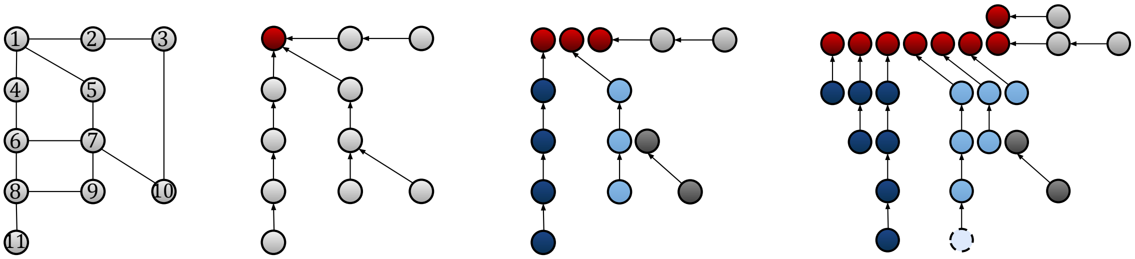
5 CRYPTANALYSIS

We now describe the theoretical limitations of what an adversary can learn from our scheme's leakage. Our scheme mitigates the attack described in [23], including for families of graphs that had resulted in full query recovery.

Leakage abuse attacks can broadly be categorised into *query recovery* attacks and *database reconstruction* attacks. In the context of GESs, the goal of query recovery is to infer the plaintext value of each issued query given the graph and the query leakage. The goal of database reconstruction is to infer the graph given the setup leakage and the query leakage of a set of k queries. Because the goal of our scheme is query privacy and we assume the graph is public, we analyze the success of an attacker attempting query recovery against our scheme. The adversary may attempt the attack using all possible queries or a subset of them. The following definitions have been adapted from [23] to follow the convention of [37].

Definition 10. *(Structural Equivalence) Let $G = (V, E)$ be a graph, and Q and Q' be sequences of SPSP queries on G of the same length. We say that Q and Q' are* ***structurally equivalent*** *if there is a permutation $\pi : V^2 \rightarrow V^2$ such that $\mathcal{L}_\mathsf{Q}(G, \pi(Q)) = \mathcal{L}_\mathsf{Q}(G, Q')$, where $\pi(Q)$ replaces each query q of sequence Q with $\pi(q)$.*

We say that π is ***consistent with the leakage*** $\mathcal{L}_\mathsf{Q}(G, Q)$.



(a) Original graph G.

(b) SDSP Tree T_1.

(c) Decomposition of T_1.

(d) Our scheme's leakage.

Figure 3: (a) Graph G, (b) its SDSP tree rooted at vertex 1, and (c) the SDSP tree decomposed into edge-disjoint paths. The GKT scheme leaks trees that are isomorphic to the SDSP trees. In contrast, our scheme only leaks edge-disjoint paths. In particular, it leaks (d) the set of minimal-length canonical fragments comprising the queried path. These fragments may contain padding to pad lengths up to the next largest power of two; padding vertices are depicted with a dotted border. Distinct queries may correspond to the same fragments.

In other words, the sequence of queries $\pi(Q)$ could have produced the observed leakage.

Definition 11. *(QR) Let $G = (V, E)$ be a graph, Q be a sequence of SPSP queries, and Π be the set of all permutations consistent with $\mathcal{L}(G, Q)$. The adversary achieves* ***query recovery (QR)*** *when it computes and outputs a mapping: $q \mapsto \{\pi(q) : \pi \in \Pi\}$ for all $q \in V^2$.*

The goal of query recovery is to compute the set of all possible queries for each query issued by the client.

Definition 12. *(Query reconstruction space) Let $G = (V, E)$ be a graph and Q be a sequence of SPSP queries. The* ***query reconstruction space*** *of $\mathcal{L}(G, Q)$ is the set of all permutations, Π, consistent with $\mathcal{L}(G, Q)$. The* ***size of the query reconstruction space*** *is $|\Pi|$.*

We denote the query reconstruction space of a graph encryption scheme GES with respect to a graph G and query sequence Q as $\mathrm{QRS}_{\mathrm{GES}}(G, Q)$.

5.1 Comparing Leakage Functions

A natural question is to ask whether PathGES "leaks less" than GKT. Bost and Fouque [4, 5] introduced the notion of order relation on leakage which states that a leakage function $\mathcal{L}$ leaks less than $\mathcal{L}'$, denoted $\mathcal{L} \preceq_Q \mathcal{L}'$, if and only if for any sequence of queries the output of $\mathcal{L}$ can be simulated using the output of $\mathcal{L}'$. See [4, 5] for a formal statement; a similar notion was independently introduced by Kamara et al. [32].

Bost and Fouque note that $\preceq$ is a partial order on the set of leakage functions. Many leakage functions are not directly comparable using this relation, as is the case for our scheme and GKT. Recall that the GKT scheme leaks "query trees" that are isomorphic to the plaintext SDSP trees. If only a subgraph of the query tree has been observed (i.e. only a fraction of SPSP queries have been issued), then one cannot compute an HLD decomposition of the tree and transform the GKT's leakage into the leakage of PathGES.

Conversely, our scheme dis-associates paths that do not share an edge, so the full query tree is often not recoverable using the observed leakage. Depending on the set of queries issued and the decomposition algorithm used, our scheme may, however, leak some information that the GKT scheme does not. For example, if the server observes a response comprising of two encrypted fragments each of size one, then it can infer that this path of length 2 is connected to a larger subtree; this is a consequence of HLD. The leakage of our scheme is thus, in part, a function of the chosen decomposition algorithm. Different tree decomposition algorithms would provide different security and efficiency trade-offs. Exploring this space will be interesting future work.

Although we cannot transform the leakage of GKT to that of PathGES for *any* sequence of queries, we can prove a weaker form of the statement when all possible queries have been issued. Many prior attacks assume that all possible queries have been issued (e.g., [21, 23, 30, 35]) and we believe that proving our scheme's leakage is less than that of GKT's in the setting when all queries have been issued is an important first step. Below, we introduce a generalization of the order relation on leakage functions [5].

Definition 13. *Define two leakage functions for a GES scheme as $\mathcal{L} = (\mathcal{L}_{\mathrm{S}}, \mathcal{L}_{\mathrm{Q}})$ and $\mathcal{L}' = (\mathcal{L}'_{\mathrm{S}}, \mathcal{L}'_{\mathrm{Q}})$. We say that $\mathcal{L}$* ***leaks less than $\mathcal{L}'$ under a family of query sequences $\boldsymbol{Q}$*** *(denoted $\mathcal{L} \preceq_{\boldsymbol{Q}} \mathcal{L}'$) if there exists a pair of polynomial-time algorithms $\mathcal{T} = (\mathsf{TSetup}, \mathsf{Tquery})$, such that, for any graph G and any query sequence $Q \in \boldsymbol{Q}$,*

- *$\mathcal{L}_{\mathrm{S}}(G) = \mathsf{TSetup}(\mathcal{L}'_{\mathrm{S}}(G))$ and*
- *$\mathcal{L}_{\mathrm{Q}}(Q) = \mathsf{Tquery}(\mathcal{L}'_{\mathrm{Q}}(G, Q))$.*

Relation $\preceq_{\boldsymbol{Q}}$ is equivalent to relation $\preceq$ in [5] when $\boldsymbol{Q}$ is the family of all query sequences.

Lemma 14. *Let $\boldsymbol{Q}$ be a family of query sequences. Relation $\preceq_{\boldsymbol{Q}}$ is a partial ordering on the set of leakage functions with respect to $\boldsymbol{Q}$.*

Proof. In order to prove that $\preceq_{\boldsymbol{Q}}$ is an order relation, we must show that the following three properties hold: (i) reflexivity, (ii) antisymmetry, and (iii) transitivity.

It is easy to see that (i) holds. That is, for any leakage function $\mathcal{L}$, we have that $\mathcal{L} \preceq_{\boldsymbol{Q}} \mathcal{L}$. One can define the trivial algorithm that takes as input $\mathcal{L}$ and outputs the same function.

For a contradiction suppose that $\mathcal{L} \preceq_{\boldsymbol{Q}} \mathcal{L}'$ and $\mathcal{L}' \preceq_{\boldsymbol{Q}} \mathcal{L}$, but that $\mathcal{L} \neq \mathcal{L}'$ with respect to $\boldsymbol{Q}$. Then there exists some graph G and some query sequence $Q \in \boldsymbol{Q}$ such that, without loss of generality, no pair of stateful poly-time algorithms $\mathcal{T}$ can simulate $\mathcal{L}_{\mathrm{S}}(G)$ using $\mathcal{L}'_{\mathrm{S}}(G)$ (respecitively, $\mathcal{L}_{\mathrm{Q}}(G, Q)$ using $\mathcal{L}'_{\mathrm{Q}}(G, Q)$, respectively. But this implies that $\mathcal{L} \npreceq_{\boldsymbol{Q}} \mathcal{L}'$. Thus property (ii) must hold.

To show that property (iii) holds, assume that $\mathcal{L} \preceq_{\mathcal{Q}} \mathcal{L}'$ and $\mathcal{L}' \preceq_{\mathcal{Q}} \mathcal{L}''$ for some functions $\mathcal{L}, \mathcal{L}'$, and $\mathcal{L}''$. By Definition 13, there exist poly-time algorithms $\mathcal{T}$ and $\mathcal{T}'$ that transform $\mathcal{L}'$ to $\mathcal{L}$ and $\mathcal{L}''$ to $\mathcal{L}'$, respectively. Define a new algorithm $\mathcal{T}' \circ \mathcal{T}$ which takes as input the leakage $\mathcal{L}''$ and outputs $\mathcal{L}$. Since both $\mathcal{T}$ and $\mathcal{T}'$ are poly-time, then so is their composition. □

Given any family of query sequences $\mathcal{Q}$, the order $\preceq_{\mathcal{Q}}$ between leakage functions implies a reverse size relation between the corresponding query reconstruction spaces. This theorem extends to database reconstruction and to all structured encryption schemes, but we state it with respect to the query reconstruction space of GESs since that is the focus of this work.

Theorem 15. *Let* GES *and* GES$'$ *be GESs with leakage functions $\mathcal{L}$ and $\mathcal{L}'$, respectively. Let $\mathcal{Q}$ be a family of query sequences. If $\mathcal{L} \preceq_{\mathcal{Q}} \mathcal{L}'$, then for all $Q \in \mathcal{Q}$*

$$|\mathsf{QRS}_{\mathsf{GES}}(G, Q)| \geq |\mathsf{QRS}_{\mathsf{GES}'}(G, Q)|.$$

Proof. (Sketch) Let $Q \in \mathcal{Q}$. Suppose for a contradiction, that there exists some assignment $\pi : V^2 \to V^2$ that is consistent with leakage $\mathcal{L}(G, Q)$ and not $\mathcal{L}'(G, Q)$, i.e., $\pi \in \mathsf{QRS}_{\mathsf{GES}}(G, Q)$ and $\pi \notin \mathsf{QRS}_{\mathsf{GES}'}(G, Q)$. But this implies that there is some additional information leaked by $\mathcal{L}$ not leaked by $\mathcal{L}'$ that cannot be derived from $\mathcal{L}'$ hence contradicting the existence of a pair of algorithms $\mathcal{T}$ that transforms $\mathcal{L}'$ to $\mathcal{L}$. Thus, we have $\pi \in \mathsf{QRS}_{\mathsf{GES}'}(G, Q)$ and the theorem follows. □

An upper bound on the leakage of PathGES. We now formally describe the leakage of the GKT scheme. We assume that the GKT scheme is instantiated using a response-revealing EMM scheme that is (Size, (QP, AP))-secure. At setup, the GKT scheme leaks the number of pairs of vertices that are connected:

$$\mathcal{L}_{\mathsf{S}}^{\mathsf{GKT}}(G) = |\{(u, v) \in V \times V \ : \ u \text{ and } v \text{ connected in } G\}|$$

For connected graphs, this is equal to the number of vertices. At query time, the scheme leaks the query pattern (QP), the length of the shortest path t, and the path intersection, denoted as PIP (as a consequence of the AP). In words, PIP reveals to the server the edges that intersect along the shortest path. See [26] for a more detailed description of the leakage. All together, we have that

$$\mathcal{L}_{\mathsf{Q}}^{\mathsf{GKT}}(G, (q_1, \ldots, q_k)) = (\mathsf{QP}, \mathsf{PIP}, \mathbf{t})$$

where $\mathbf{t} = (t_1, \ldots, t_k)$ is a vector and t_i is the length of the shortest path returned in response to query q_i.

Theorem 16. *Let G be a connected graph and let $\mathcal{Q}$ be the family of query sequences in which each SPSP query is issued at least once. Let $\mathcal{L}^{\mathsf{GKT}}$ and $\mathcal{L}^{\mathsf{PathGES}}$ denote the leakage functions of* GKT *[26] and* PathGES *(Figure 2), respectively. Then $\mathcal{L}^{\mathsf{PathGES}} \preceq_{\mathcal{Q}} \mathcal{L}^{\mathsf{GKT}}$.*

Given the query leakage of the GKT scheme when all queries have been issued, one can construct n query trees $\{S_i\}_{i \in [n]}$. There exists a 1-1 correspondence between $\{S_i\}_{i \in [n]}$ and the SDSP trees $\{T_i\}_{i \in [n]}$ such that each pair of trees is isomorphic [23]. Our transformer $\mathcal{T}$ thus takes as input the leakage of GKT and computes these query trees. From here, it can compute the edge-disjoint path decomposition of each tree and the set of fragments for each path in the decomposition. The proof can be found in Appendix B.6.

Corollary 17. *Let G be a connected graph and $\mathcal{Q}$ be the family of query sequences such that each SPSP query appears at least once. Then for any sequence $Q \in \mathcal{Q}$, it holds that $|\mathsf{QRS}_{\mathsf{PathGES}}(G, Q)| \geq |\mathsf{QRS}_{\mathsf{GKT}}(G, Q)|$.*

Since many schemes are not comparable under $\preceq$, Kornaropoulos et al. [37] propose *leakage inversion* as a way to quantify the privacy of schemes. They show how the size of the reconstruction space and the distance of its members from the original plaintext database are good metrics for quantifying scheme privacy. In the following section, we characterize the query reconstruction space of our PathGES scheme and show that its size is super-polynomially larger than that of the GKT scheme for various graph families.

5.2 QR from the GKT scheme's leakage

In remainder of this section, we assume that every possible query in $V \times V$ has been issued once. This represents the strongest passive adversary without auxiliary information. Yet even in this strong setting, we demonstrate an information theoretical gap between what the adversary can reconstruct from the leakage of PathGES versus that of the GKT scheme.

Recall that if all possible SPSP queries are issued to the GKT scheme, then an adversary can construct a set of n query trees; these query trees are one-to-one with the SDSP trees. Each SDSP tree is rooted at a vertex in V and the paths correspond to the shortest paths whose destination is the root. Thus, queries can be recovered up to the possible isomorphisms that exist between the query trees and the SDSP trees. This notion is formalized as follows.

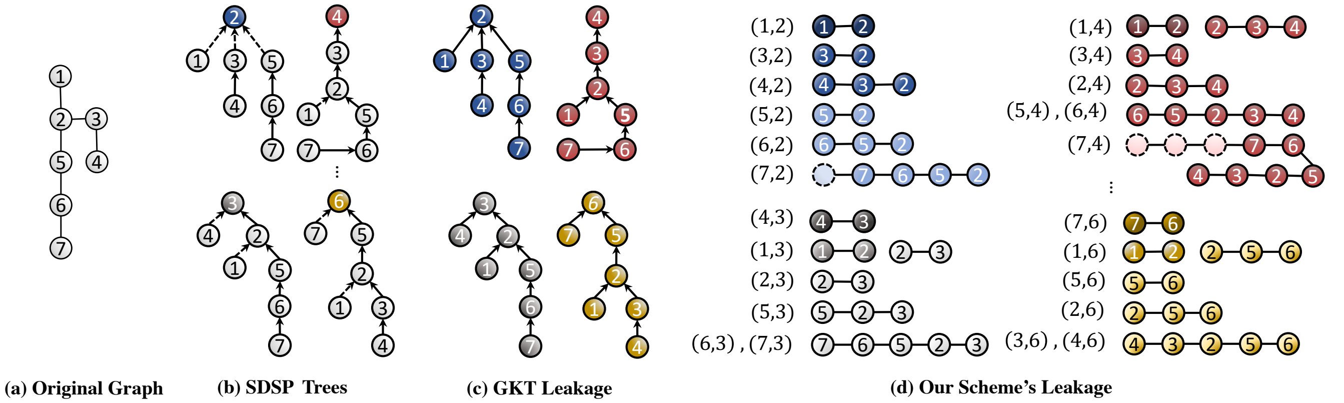
Lemma 18 ([23]). *Let $G = (V, E)$ be a graph encrypted using* GKT, *$\{T_r\}_{r \in V}$ be the SDSP trees of G, Q be any sequence in which each SPSP query is issued once, and $q = (u, v)$ be a query in Q. If there exists a vertex $w \in V$ such that there is a rooted tree isomorphism $\phi : T_v \to T_w$, then there exists an assignment $\pi : V^2 \to V^2$ consistent with the leakage $\mathcal{L}_{\mathsf{Q}}(G, Q)$ such that $\pi(q) = (\phi(u), w)$.*

What this theorem tells us, is that queries can be recovered up to the possible isomorphisms between the query trees and the SDSP trees. If there only exists one possible matching between the query trees and the SDSP trees *and* there only exists one isomorphism between each pair of trees, then queries can be uniquely recovered. This strong form of recovery is called **full query recovery (FQR)** [23]. Falzon and Paterson note that there exist families of trees for which FQR is always possible. One such family is the family of graphs that have one central vertex v and paths of distinct lengths incident to v. Figure 4 depicts such a graph.

5.3 QR from the PathGES scheme's leakage

In contrast to the GKT scheme, PathGES decomposes each SDSP tree into edge-disjoint paths before encrypting the paths using a response hiding EMM. As a result, an adversary cannot necessarily associate the leakage of SPSP queries with the same destination vertex whose paths are edge disjoint. Specifically, for each SDSP tree T_r, an adversary can at most recover the queries up to isomorphism of the trees rooted at the *children* of r.

For a concrete example, consider the graph in Figure 4 and the leakage resulting from the SPSP queries $(1, 2)$, $(5, 2)$, and $(4, 3)$. The GKT scheme leaks the fact that paths $p_{1,2}$ and $p_{5,2}$ share the same



(a) Original Graph **(b) SDSP Trees** **(c) GKT Leakage** **(d) Our Scheme's Leakage**

Figure 4: A comparison of the leakage of GKT **versus** PathGES**. The attack in [23] against the (a) original graph results in full query recovery i.e., there exists a single isomorphism between each (b) SDSP tree and the (c) query trees computed from the GKT scheme's leakage. Thus each query can be uniquely recovered. In contrast, our scheme results in numerous fragments, with distinct queries potentially returning the same fragment. For example, queries** $(6, 3)$ **and** $(7, 3)$ **result in the same response and hence cannot be distinguished.**

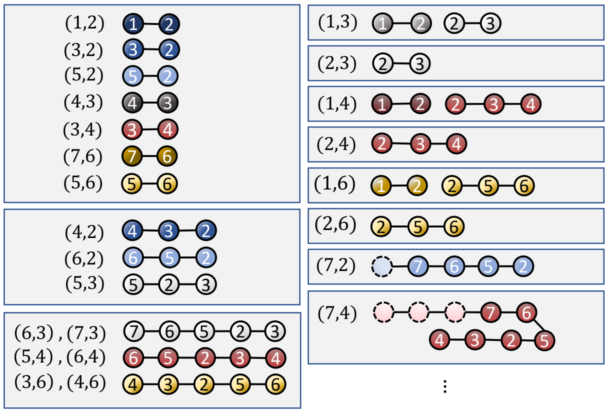


Figure 5: A partition of indistinguishable queries when encrypting the graph in Figure 4(a) with our scheme. The leakage induced by our scheme is depicted in Figure 4(d).

destination vertex and that path $p_{4,3}$ has a different destination. In contrast, our scheme does not leak anything beyond the lengths of their respective fragments and which fragments appear together. All three queries result in fragments of the same length and these fragments cannot be distinguished.

We now formally prove that QR is only possible up to isomorphisms of subtrees rooted at the children of the SDSP trees' roots.

Lemma 19. *Let* $G = (V, E)$ *be a graph encrypted using* PathGES, $\{T_r\}_{r \in V}$ *be the SDSP trees of* G, Q *be any sequence in which each SPSP query was issued once, and* $q = (u, v)$ *be a query in* Q*. If there exists a vertex* $w \in V$ *and children* c *and* d *of the roots in* T_v *and* T_w*, respectively, such that there is a rooted tree isomorphism* $\phi : T_v[c] \cup (c, v) \longrightarrow T_w[d] \cup (d, w)$*, then there exists an edge-disjoint path decomposition of* $\{T_r\}_{r \in V}$ *and assignment* $\pi : V^2 \longrightarrow V^2$ *consistent with the leakage* $\mathcal{L}_Q(G, Q)$ *such that* $\pi(q) = (\phi(u), w)$*.*

The proof can be found in Appendix B.7.

If there are two isomorphic subtrees $T_v[c]$ and $T_w[d]$, then there is a set of queries of the form (a, b), $a \in T_v[c]$ that is indistinguishable from a set of queries (a', b'), $a' \in T_w[d]$. In other words, if there exists two isomorphic subtrees from non-isomorphic SDSP trees, the queries associated with these subtrees cannot be distinguished. In contrast, the queries would be trivially distinguishable given leakage from the GKT scheme since the query trees would be non-isomorphic. Thus, many queries that would otherwise be uniquely recoverable when using the GKT scheme, cannot be uniquely identified when using PathGES.

One important example is the asymmetric star which we recall in Figure 4; whereas full query recovery can be achieved using the leakage from GKT, our scheme's leakage results in several indistinguishable queries and strictly less leakage (see Figure 5).

The implication of Lemma 19 is that, in many cases, it is not possible to construct complete query trees. This is especially true in graphs in which the roots of the SDSP trees and their descendents induce isomorphic subtrees. Recall, that the first step of the attack in [23] is to construct the complete query trees from the query leakage of the GKT scheme. Each query tree is isomorphic to some spanning shortest path tree in G and, importantly, computing such isomorphisms between rooted trees can be done so efficiently. In contrast to GKT, our use of edge-disjoint fragments ensures that PathGES does not leak the entire query tree except in extreme cases (e.g., the complete graph on 2 nodes). Instead, it leaks a collection of fragments, many of which are computationally indistinguishable without auxiliary knowledge (Figure 5).

5.4 Reconstruction Space

We now make some more general observations about the reconstruction space of our scheme.

Lemma 20. *Let* $G = (V, E)$*, and fix a set of SDSP trees for* G *and a decomposition of each tree into canonical fragments. For each canonical fragmentof length* L*, there are at least* $L/2$ *equivalent queries.*

Proof. Let $v \in V$ and f be a fragment in SDSP tree T_v. Let the corresponding path of f in T_v be $(u_0, u_2, \dots, u_{k^2})$ where u_{k^2} is the node closest to the root v. Then the set of queries $\{(u_i, v) : 0 \leq i \leq k^2/2\}$ result in the same response. □

Corollary 21. *Let F be the set of all fragments. Then the reconstruction space is of size at least $\prod_{f \in F, |f| \geq 2} (|f|/2)!$*

For concreteness, we compute the sizes of the reconstruction spaces of the GKT scheme and PathGES for the following graph families: (1) K_n, the complete graph on n vertices, (2) L_n, the line graph on n vertices, (3) S_n, the "asymmetric star" with one central vertex and $\log(n+1)-1$ incident paths of lengths $2^1, \ldots, 2^{\log(n+1)-1}$, and (4) G_n, the grid graph of size $\sqrt{n} \times \sqrt{n}$. The results are summarized in Table 1 and in Theorems 28 and 29. The reconstruction space sizes for the GKT scheme are exact or upper bounds, and the sizes for our scheme may be lowerbounds. For all four graph families, the gap of the query reconstruction spaces is of size at least super-polynomial in n. Thus, when all queries are issued, PathGES leaks *strictly less* than GKT for these families.

We note that our cryptanalysis and results in Table 1 assume that all queries have been issued at least once. However, notice that Lemma 20 and Corollary 21 apply even when fewer than all queries are issued. In particular, unlike the GKT scheme, distinct SPSP queries may result in the same response, thus making query recovery more challenging.

The above discussion demonstrates that QR is a challenging problem for many graphs and we now explain why we believe this problem to be hard even for general graphs. The query reconstruction space can be characterized by identifying queries with equivalent leakage. To do this, we introduce the notion of a **closure graph** C for leakage $\mathcal{L}(G, Q)$, where G is a graph and Q is a sequence containing each SPSP query. We note that the assumption of observing all queries implies a strong adversary in a contrived and unrealistic scenario; however, if even a strong adversary cannot succeed, it suggests that a real world adversary cannot succeed since it cannot observe as many queries.

Let the vertex set of C be the set of fragment identifiers F observed in the leakage. The pair $(f, f') \in F \times F$ is an edge in C if and only if the fragments f and f' appear together in a response. Each vertex $f \in F$ is labeled with the length of the fragment f. Each edge (f, f') is labeled with the number of distinct queries that the two fragments appear in together.

Observe that this graph encodes information about what fragments appear together and whether two fragments f, f' both appear in some response with another fragment f'' (even if they don't directly appear in a response together). Two queries are distinguishable if and only if their fragments appear in non-isomorphic components of the closure graph (here, we consider isomorphisms that not only preserve edges, but also vertex and edge weights). For a concrete example, consider queries $(2, 6)$ and $(5, 3)$ in Figure 4, and their respective fragments f_1 and f_2. Although both f_1 and f_2 are of length 2, the fragment f_1 is also returned as part of the response of query $(1, 6)$. Thus, f_1 and f_2 belong to non-isomorphic components of the closure graph and they are thus distinguishable.

To compute the query reconstruction space using this approach, an adversary must (1) compute the closure graph from the plaintext graph, (2) construct the closure graph from the leakage, and (3) find all possible isomorphisms between the two closure graphs and derive the partition of equivalent queries. The graph isomorphism problem is not known to be solvable in polynomial time for general graphs, and the problem of determining whether an isomorphism that preserves edges weights and vertex labels exists between two graphs is at least as hard as the standard graph isomorphism problem. Generalizing further to the case where only a subset of queries is observed or certain queries are issued multiple times, the problem becomes one of finding subgraph isomorphisms.

Dataset	Graph Characteristics					
	$\|V\|$	$\|E\|$	d	# Comp.	Dia-meter	Max Frag Length
InternetRouting	35	323	0.543	1	2	2
CA-GrQc	46	1030	0.995	1	2	2
email-Eu-core	1005	16,706	0.0331	20	7	4
facebook	4039	88,234	0.011	1	8	4
p2p-Gnutella08	6301	20,777	0.001	2	9	8
p2p-Gnutella04	10,876	39,994	0.0006	1	10	8
Swiss	19,976	24,009	0.00012	1	311	512
Cali	21,693	21,693	0.00009	2	491	512

Table 2: Details about the real-world datasets used in our experiments. $|V|$ denotes the number of vertices, $|E|$ the number of edges, and $d = 2|E|/(|V|^2 - |V|)$ the density of the graph.

We expect closure graphs to be dense non-trees, since fragments that appear together in the response of an SPSP query are in one-to-one correspondence with nodes in the closure graph that induce a clique. The graph isomorphism problem is a long standing open question and seeing how graph isomorphism algorithms perform on closure graphs of real world data sets is interesting future work.

6 EMPIRICAL EVALUATION

We now evaluate our scheme's performance on real world datasets and compare its performance to the GKT scheme [26].

Experimental setup. We implemented our scheme using Python 3.8.10 and ran our experiments on a compute cluster with 2U Rackmount Chassis, 64 Core AMD EPYC 7742 2.25GHz Processor, and 512GB DDR4 3200MHz ECC Server Memory. For comparison, we implemented the GKT scheme. Both implementations used the same compute node for the client and the server, so our results do not include network latency.

Due to memory constraints imposed by the compute cluster, we wrote the plaintext and encrypted multimaps out to SQLite databases. Writing the EMMs to disk incurs significant time overhead compared to storing everything in main memory, but we believe this is a better reflection of what would happen in practice for realistic work loads.

Graphs. We used the NetworkX library version 3.1 [19] to represent and manipulate graphs. We used our own implementation of the HLD algorithm (Figure 8).

Cryptographic primitives. The cryptographic primitives were implemented using the Cryptography library version 42.0.5 [58]. For symmetric encryption we used AES in CBC mode with 16B block size and key length; for cryptographic hash functions we used SHA-256; for search token generation we used HMAC with SHA-256. We implemented the encrypted EMMs of PathGES using the response-revealing scheme of [9] for EMM-RR and a straightforward response-hiding modification of this scheme for EMM-RH. Encryption was parallelized across 20 cores.

Dataset	GKT				PathGES							
	M Size	**EM** Size	Encryption Time	Total Time	M_1 Size	M_1 % Pad	$\mathbf{EM}_1$ Size	M_2 Size	M_2 % Pad	$\mathbf{EM}_2$ Size	Encryption Time	Total Time
InternetRouting	102.4KB	180KB	93.9ms	344ms	147KB	87.3	213KB	86KB	30.4	369KB	264ms	442ms
CA-GrQc	172KB	303KB	100ms	423ms	246KB	85.3	365KB	147KB	30.2	631KB	312ms	537ms
email-Eu-core	79.6MB	138MB	10.1s	21.8s	188MB	64.8	244MB	84MB	29.8	299MB	40.2s	64.5s
facebook	1.39GB	2.31GB	2.94mins	5.07mins	3.64GB	60.7	4.47GB	1.53GB	30.2	4.87GB	11.8min	17.8min
p2p-Gnutella08	3.38GB	5.62GB	6.85mins	11.5mins	9.59GB	61.0	11.6GB	3.76GB	31.5	11.9GB	29.7mins	43.4mins
p2p-Gnutella04	10.1GB	16.8GB	20.5mins	34.7mins	30.4GB	58.3	37.4GB	11.7GB	31.6	35.6GB	1.70hr	2.42hr
Swiss	34.5GB	57.2GB	1.19hr	2.05hr	109GB	54.9	137GB	31.4GB	34.8	121GB	4.90hr	6.57hr
Cali	40.7GB	67.6GB	1.44 hr	2.80hr	129GB	53.6	161GB	32.4GB	32.7	143GB	5.56hr	8.65hr

Table 3: Setup results for both GKT **and** PathGES**.**

6.1 Datasets

We evaluated our scheme on the same social network datasets as Ghosh et al. [26] and Falzon and Paterson [23], along with two geographical datasets: the Swiss Federal Railway timetable [59] and the California road network [43]. See Table 2 for more details.

- **InternetRouting [42]:** A dataset from the University of Oregon Route Views Project. A dense subgraph ($n = 35$) was extracted using the dense subset extraction algorithm by Charikar [13] as implemented by Ambavi et al. [2].
- **Ca-GrQc [42]:** A network of the General Relativity and Quantum Cosmology arXiv collaborations from January 1993 to April 2003. A subgraph ($n = 46$) was extracted using dense subset extraction.
- **email-EU-core [42]:** A network of internal emails sent between members of a large European research institution. We parsed the data as a non-directed graph, i.e. an edge (u, v) exists if either u sent v an email or vice versa.
- **facebook [42]:** A social network derived from Facebook friends lists; it includes all edges from the ego networks collected.
- **p2p-Gnutella [42]:** Two datatsets depicting the Gnutella peer-to-peer network from August 4 and 8 2002.
- **Swiss [59]:** A timetable of the Swiss Railway from December 13, 2015 to December 10, 2016 parsed as a graph [48]. We extracted the largest component from this graph.
- **Cali [43]:** A dataset of the California Road Network. It was used in prior works on range schemes e.g. [22, 49].

6.2 Performance

Setup. Setup comprises of the KeyGen and Encrypt algorithms. We report setup benchmarks in Table 2 including sizes of the plaintext and encrypted multimaps, and total setup time. Sizes of the multimaps were obtained using the `os.path.getsize` function to measure the size of the corresponding database files.

Setup times were practical, ranging from as little as 442 ms ($n = 35$) to 8.65 hours ($n = 21,693$). Client-side storage required only a 32B key: one 16B key for each of the two EMMs. Similar to the GKT scheme, server-side storage is the most significant cost. Total size of the encrypted database ranged from 582KB ($n = 35$) to 304GB ($n = 21,693$). Due to how the edges were stored and encryption overhead, the sizes of $\mathbf{EM}_1$ and $\mathbf{EM}_2$ were at most 1.48 times and 4.41 times larger than the plaintext multimaps, respectively. Storage is proportional to the number of nodes and independent of graph density. In contrast, the GKT scheme's encryption overhead varies inversely with graph density.

Query. Querying involves executing 3 algorithms: Token (computation of the search token), Search (look-up of the encrypted records), and Reveal (decryption of the response). We depict query benchmarks in Figures 6 and 15. For each dataset we sampled 100,000 uniformly random queries, partitioned the queries based on path length (Figure 6) or number of fragments (Figure 15), and averaged the benchmarks within each set of the partition.

All three query algorithms are very efficient, measuring in at most tens of milliseconds. Experimentally, the Search algorithm's run time increases linearly with respect to both path length and number of fragments, since longer paths are likely to result in more fragments. Similarly, the run time of Reveal increases linearly with respect to both path length and number of fragments. However, we see a closer correlation to length since response decryption is only dependent on the number of edges returned. Response size varies proportionally with the length of the path, with ~100B increase for each additional vertex in the queried path.

6.3 Comparison with GKT

In Table 3, we see that GKT requires less setup time overall and that the encryption time of GKT constitutes a smaller percentage of the overall setup time. In other words, building the multimap is the more time-intensive part of setup for GKT as compared to PathGES. PathGES leaks less information about the graph structure at setup, but as a result, requires more storage overhead due to padding.

Figure 7 depicts query benchmarks for both schemes for p2p-Gnutella04 and California. We issued 100,000 uniformly random queries, partitioned the results with respect to path length, and averaged the attribute for each length. Token is very efficient for both schemes and takes approximately 0.043ms. Overall, Search runtime is faster for our scheme despite its higher theoretical complexity. The discrepancy between worst-case complexity and real-world performance can be attributed to the difference in number of calls to cryptographic primitives that our implementations of the EMM schemes must make. A call to Search.EMM-RR requires computing for each value: one hash evaluation (to obtain the encrypted label) and one symmetric decryption (to reveal the value).

In contrast, a call to Search.EMM-RH only requires evaluating one hash for each value (to obtain the encrypted label). The GKT scheme looks up t edges in a dictionary encrypted using the response-revealing scheme. Our scheme instead looks up $t + \delta$

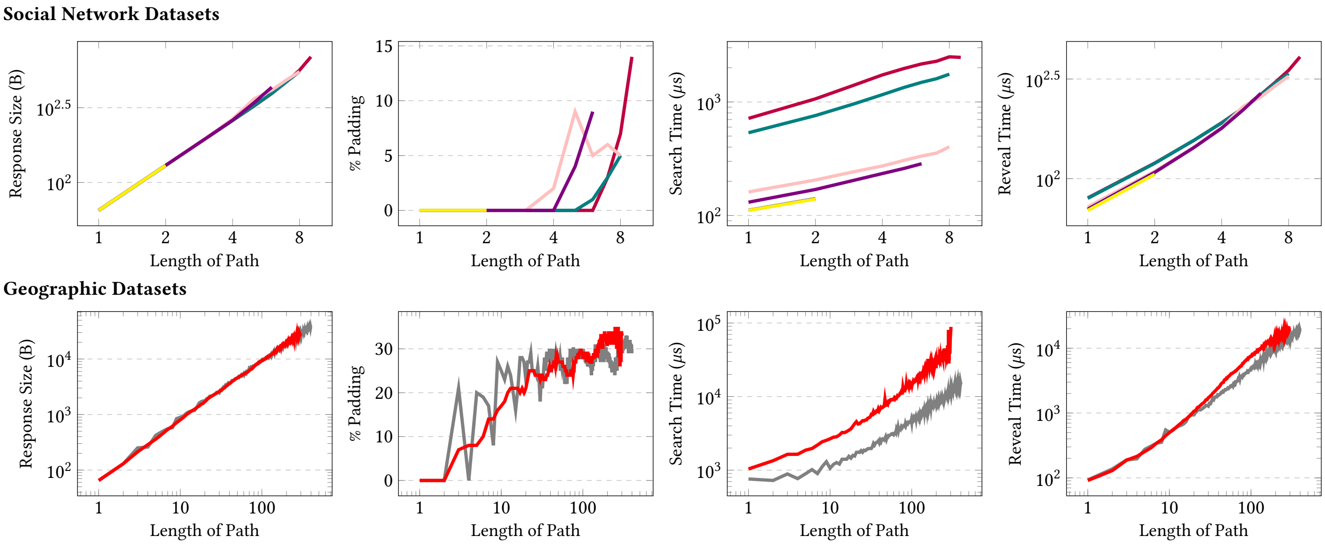


Figure 6: Query benchmarks with respect to the length of the path queried. We use the following symbols for the social network datasets: InternetRouting (▬), Ca-GrQc (▬), email-EU-core (▬), facebook (▬), p2p-Gnutella08 (▬), p2p-Gnutella04 (▬). And the following symbols for the geographic datasets: Swiss (▬) and Cali (▬). For each dataset we issued 100,000 random queries, partitioned them based on path length, and took the average of the respective attribute within each set of the partition. Since the queries were randomly sampled, we do not necessarily observe all possible path lengths.

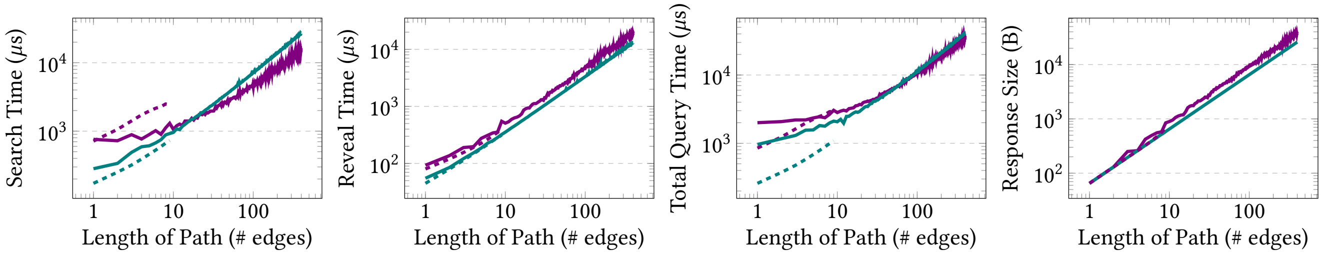


Figure 7: PathGES (▬) **and** GKT (▬) **query benchmarks for the p2p-Gnutella04 (dashed) and California (solid) datasets. Results were averaged over 100,000 uniformly random SPSP queries. For paths of length 10 or more,** Search **time takes approximately** 3× **more for** GKT. **For longer paths, we also see that the total round-trip query time is faster for** PathGES. **On the other hand,** Reveal **is faster for** GKT. **The response size is on average comparable to that of** GKT.

edges using the response-hiding scheme, $\delta \in [1, t+1]$. In practice, δ is closer to 1 for the social network graphs. These observations were confirmed using a line-profiler [36] to compute the length of time the search operation spent on each line of code. Our scheme's search time could be further decreased through parallelization. In contrast, the GKT scheme's search is intrinsically sequential.

Reveal is marginally faster for the GKT scheme, since it only involves symmetric decryption of the vertices in the shortest path. In our scheme, the returned fragments may include additional nodes and padding vertices that are not in the shortest path. Moreover, the Reveal algorithm of our response-hiding EMM implementation also entailed a key-derivation step, hence the small additive increase in decryption time required by PathGES.

Total round-trip query time is efficient and practical for both schemes. However, as the length of the path increases, the total query time of GKT overtakes that of PathGES. Response size increases linearly with respect to the path length for both schemes and both datasets. On average, the response size of PathGES almost matches that of GKT for the social network graphs, despite the worst-case 2x constant factor overhead in bandwidth.

7 CONCLUSION

We present a new graph encryption scheme for single-pair shortest path (SPSP) queries. We describe a new data structure that is designed to optimize bandwidth and query time, whilst also increasing security. Our scheme built upon this data structure achieves optimal bandwidth complexity and mitigates the attack by Falzon and Paterson [23]. We generalize the notion of what it means for one leakage function to leak less than another leakage function [5] to families of query sequences, and show that our scheme leaks less than the GKT scheme [26] for all query sequences in which each SPSP query is issued at least once. We support our scheme with a proof of security and a thorough cryptanalysis. We then demonstrate our scheme's practicality by implementing and evaluating it on a number of real-world social network and geographic datasets.

ACKNOWLEDGMENTS

This research is supported by Armasuisse Science and Technology, the U.S. National Science Foundation, and the James A. and Julie N. Brown Professorship of Computer Science at Brown University. Francesca Falzon would like to thank Kien T. Truong for help with optimizing the PathGES implementation. Francesca Falzon and Roberto Tamassia would also like to thank Lilika Markatou for insightful discussions on structured encryption schemes and leakage abuse attacks. This work was done in part while Francesca Falzon was at Brown University and at the University of Chicago.

REFERENCES

[1] Amazon. 2021. Amazon Neptune. https://aws.amazon.com/neptune/ Accessed on September 10, 2022.

[2] Heer Ambavi, Mridul Sharma, and Varun Gohil. 2020. Densest-Subgraph-Discovery. https://github.com/varungohil/Densest-Subgraph-Discovery.

[3] Raphaël Bost. 2016. Sophos: Forward Secure Searchable Encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1143–1154.

[4] Raphaël Bost. 2018. *Searchable Encryption: New Constructions of Encrypted Databases.* Université de Rennes 1.

[5] Raphael Bost and Pierre-Alain Fouque. 2019. Security-Efficiency Tradeoffs in Searchable Encryption. *Proc. Priv. Enhancing Technol.* 2019, 4 (2019), 132–151. https://doi.org/10.2478/popets-2019-0062

[6] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. 2017. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1465–1482. https://doi.org/10.1145/3133956.3133980

[7] Alina Campan, Yasmeen Alufaisan, and Traian Marius Truta. 2015. Preserving Communities in Anonymized Social Networks. *Trans. Data Privacy* 8, 1 (dec 2015), 55–87.

[8] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-Abuse Attacks Against Searchable Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 668–679. https://doi.org/10.1145/2810103.2813700

[9] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic searchable encryption in very-large databases: data structures and implementation. In *21st Annual Network and Distributed System Security Symposium 2014 (NDSS 2014).*

[10] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. 2013. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *Advances in Cryptology (CRYPTO).* Springer, Lyon, France, 353–373.

[11] David Cash, Ruth Ng, and Adam Rivkin. 2021. Improved Structured Encryption for SQL Databases via Hybrid Indexing. In *Applied Cryptography and Network Security*, Kazue Sako and Nils Ole Tippenhauer (Eds.). Springer International Publishing, Cham, 480–510.

[12] Javad Ghareh Chamani, Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2023. GraphOS: Towards Oblivious Graph Processing. *Proc. VLDB Endow.* 16, 13 (sep 2023), 4324–4338. https://doi.org/10.14778/3625054.3625067

[13] Moses Charikar. 2000. Greedy Approximation Algorithms for Finding Dense Components in a Graph. In *Approximation Algorithms for Combinatorial Optimization*, Klaus Jansen and Samir Khuller (Eds.). Springer, Berlin, Heidelberg, 84–95. https://doi.org/10.1007/3-540-44436-x_10

[14] Melissa Chase and Seny Kamara. 2010. Structured Encryption and Controlled Disclosure. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6477)*, Masayuki Abe (Ed.). Springer, 577–594.

[15] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2002. Reachability and Distance Queries via 2-Hop Labels. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, California) *(SODA '02)*. Society for Industrial and Applied Mathematics, USA, 937–946.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

[17] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (Alexandria, Virginia, USA) *(CCS '06)*. Association for Computing Machinery, New York, NY, USA, 79–88. https://doi.org/10.1145/1180405.1180417

[18] Marc Damie, Florian Hahn, and Andreas Peter. 2021. A Highly Accurate Query-Recovery Attack against Searchable Encryption using Non-Indexed Documents. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 143–160. https://www.usenix.org/conference/usenixsecurity21/presentation/damie

[19] NetworkX Developers. 2021. NetworkX. https://networkx.org/ version 2.6.2.

[20] Minxin Du, Peipei Jiang, Qian Wang, Sherman S. M. Chow, and Lingchen Zhao. 2023. Shielding Graph for eXact Analytics with SGX. *IEEE Transactions on Dependable and Secure Computing* 01 (jan 2023), 1–11. https://doi.org/10.1109/TDSC.2023.3241164

[21] Francesca Falzon, Evangelia Anna Markatou, Akshima, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tamassia. 2020. Full Database Reconstruction in Two Dimensions. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 443–460. https://doi.org/10.1145/3372297.3417275

[22] Francesca Falzon, Evangelia Anna Markatou, Zachary Espiritu, and Roberto Tamassia. 2022. Range Search over Encrypted Multi-Attribute Data. *Proc. VLDB Endow.* 16, 4, 587–600.

[23] Francesca Falzon and Kenneth G. Paterson. 2022. An Efficient Query Recovery Attack Against a Graph Encryption Scheme. In *Computer Security – ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30, 2022, Proceedings, Part I* (Copenhagen, Denmark). Springer-Verlag, Berlin, Heidelberg, 325–345. https://doi.org/10.1007/978-3-031-17140-6_16

[24] Craig Gentry and Dan Boneh. 2009. *A fully homomorphic encryption scheme.* Vol. 20:09. Stanford university Stanford.

[25] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New Constructions for Forward and Backward Private Symmetric Searchable Encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 1038–1055. https://doi.org/10.1145/3243734.3243833

[26] Esha Ghosh, Seny Kamara, and Roberto Tamassia. 2021. *Efficient Graph Encryption Scheme for Shortest Path Queries.* Association for Computing Machinery, New York, NY, USA, 516–525.

[27] Anselme Goetschmann. 2020. *Design and Analysis of Graph Encryption Schemes.* Master's Thesis. ETH Zürich.

[28] Andrew V. Goldberg. 2007. Point-to-Point Shortest Path Algorithms with Preprocessing. In *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4362)*, Jan van Leeuwen, Giuseppe F. Italiano, Wiebe van der Hoek, Christoph Meinel, Harald Sack, and Frantisek Plásil (Eds.). Springer, 88–102. https://doi.org/10.1007/978-3-540-69507-3_6

[29] Oded Goldreich. 1987. Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (New York, New York, USA) *(STOC '87)*. Association for Computing Machinery, New York, NY, USA, 182–194.

[30] Paul Grubbs, Marie-Sarah Lacharite, Brice Minaud, and Kenneth G. Paterson. 2018. Pump up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 315–331. https://doi.org/10.1145/3243734.3243864

[31] Seny Kamara and Tarik Moataz. 2018. SQL on Structurally-Encrypted Databases. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11272)*, Thomas Peyrin and Steven D. Galbraith (Eds.). Springer, 149–180.

[32] Seny Kamara, Tarik Moataz, and Olya Ohrimenko. 2018. Structured Encryption and Leakage Suppression. In *Advances in Cryptology – CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 339–370. https://doi.org/10.1007/978-3-319-96884-1_12

[33] Seny Kamara and Charalampos Papamanthou. 2013. Parallel and Dynamic Searchable Symmetric Encryption. In *Financial Cryptography and Data Security*, Ahmad-Reza Sadeghi (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 258–274.

[34] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic Searchable Symmetric Encryption. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 965–976. https://doi.org/10.1145/2382196.2382298

[35] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *Proceedings of the 2016 ACM SIGSAC*

Conference on Computer and Communications Security (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1329–1340. https://doi.org/10.1145/2976749.2978386

[36] Robert Kern. 2023. line-profiler. https://pypi.org/project/line-profiler/ version4.0.3.

[37] Evgenios M. Kornaropoulos, Nathaniel Moyer, Charalampos Papamanthou, and Alexandros Psomas. 2022. Leakage Inversion: Towards Quantifying Privacy in Searchable Encryption. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 1829–1842. https://doi.org/10.1145/3548606.3560593

[38] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2020. The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution. In *Proceedings IEEE Symposium on Security and Privacy (S&P)*.

[39] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2021. Response-Hiding Encrypted Ranges: Revisiting Security via Parametrized Leakage-Abuse Attacks. In *Proceedings IEEE Symposium on Security and Privacy (S&P)*.

[40] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2018. Improved reconstruction attacks on encrypted data using range query leakage. In *Proceedings of the IEEE Symposium on Security and Privacy 2018 (S&P 2018)*.

[41] Russell W. F. Lai and Sherman S. M. Chow. 2017. Forward-Secure Searchable Encryption on Labeled Bipartite Graphs. In *Applied Cryptography and Network Security*, Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi (Eds.). Springer International Publishing, Cham, 478–497.

[42] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[43] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. 2005. On Trip Planning Queries in Spatial Databases. In *Advances in Spatial and Temporal Databases*, Claudia Bauzer Medeiros, Max J. Egenhofer, and Elisa Bertino (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 273–290.

[44] Mingyue Li, Chunfu Jia, Ruizhong Du, and Wei Shao. 2021. Forward and Backward Secure Searchable Encryption Scheme Supporting Conjunctive Queries over Bipartite Graphs. *IEEE Transactions on Cloud Computing* (2021), 1–1. https://doi.org/10.1109/TCC.2021.3131176

[45] Mingyue Li, Chunfu Jia, Ruizhong Du, and Wei Shao. 2021. Forward and Backward Secure Searchable Encryption Scheme Supporting Conjunctive Queries over Bipartite Graphs. *IEEE Transactions on Cloud Computing* (2021), 1–1. https://doi.org/10.1109/TCC.2021.3131176

[46] Chang Liu, Liehuang Zhu, and Jinjun Chen. 2017. Graph Encryption for Top-K Nearest Keyword Search Queries on Cloud. *IEEE Transactions on Sustainable Computing* 2, 4 (2017), 371–381. https://doi.org/10.1109/TSUSC.2017.2704163

[47] Chang Liu, Liehuang Zhu, Xiangjian He, and Jinjun Chen. 2021. Enabling Privacy-Preserving Shortest Distance Queries on Encrypted Graph Data. *IEEE Trans. Dependable Secur. Comput.* 18, 1 (jan 2021), 192–204. https://doi.org/10.1109/TDSC.2018.2880981

[48] Kevin Scott Mader. 2019. Parsing SBB Routes as a Graph. https://www.kaggle.com/code/kmader/parsing-sbb-routes-as-a-graph/notebook.

[49] Evangelia Anna Markatou, Francesca Falzon, Roberto Tamassia, and William Schor. 2021. Reconstructing with Less: Leakage Abuse Attacks in Two Dimensions. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2243–2261. https://doi.org/10.1145/3460120.3484552

[50] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. 2015. GRECS: Graph Encryption for Approximate Shortest Distance Queries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 504–517.

[51] Kyriakos Mouratidis and Man Lung Yiu. 2012. Shortest Path Computation with No Information Leakage. *Proceedings of the VLDB Endowment* 5, 8 (2012), 692–703.

[52] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. 2014. Dynamic Searchable Encryption via Blind Storage. In *2014 IEEE Symposium on Security and Privacy*. 639–654.

[53] Inc. Neo4j. 2021. Neo4j. https://neo4j.com/ Accessed on September 10, 2021.

[54] Ruth Ng, Alexander Hoover, David Cash, and Eileen Ee. 2023. Structured Encryption for Indirect Addressing. *IACR Cryptol. ePrint Arch.* (2023), 1146. https://eprint.iacr.org/2023/1146

[55] Ontotext. 2021. GraphDB. https://graphdb.ontotext.com/ Accessed on September 10, 2022.

[56] Simon Oya and Florian Kerschbaum. 2021. Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 127–142. https://www.usenix.org/conference/usenixsecurity21/presentation/oya

[57] Geong Sen Poh, Moesfa Soeheila Mohamad, and Muhammad Reza Z'aba. 2012. Structured Encryption for Conceptual Graphs. In *Advances in Information and Computer Security*, Goichiro Hanaoka and Toshihiro Yamauchi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 105–122.

[58] Python Cryptographic Authority. 2023. pyca/cryptography. https://cryptography.io/en/latest/ version 39.0.0.

[59] Open-Data-Plattform öV Schweiz. 2016. Fahrplan 2016 (GTFS). https://opentransportdata.swiss/en/dataset/timetable-2016-gtfs.

[60] Adam Sealfon. 2016. Shortest Paths and Distances with Differential Privacy. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (San Francisco, California, USA) *(PODS '16)*. Association for Computing Machinery, New York, NY, USA, 29–41.

[61] Meng Shen, Baoli Ma, Liehuang Zhu, Rashid Mijumbi, Xiaojiang Du, and Jiankun Hu. 2018. Cloud-Based Approximate Constrained Shortest Distance Queries Over Encrypted Graphs With Privacy Protection. *IEEE Transactions on Information Forensics and Security* 13, 4 (2018), 940–953. https://doi.org/10.1109/TIFS.2017.2774451

[62] Daniel D. Sleator and Robert Endre Tarjan. 1983. A Data Structure for Dynamic Trees. *J. Comput. Syst. Sci.* 26, 3 (jun 1983), 362–391.

[63] Dawn Song, David Wagner, and Adrian Perrig. 2000. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*. 44–55.

[64] Venkateshwaran Venkataramani, Zach Amsden, Nathan Bronson, George Cabrera III, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Jeremy Hoon, Sachin Kulkarni, Nathan Lawrence, Mark Marchukov, Dmitri Petrov, and Lovro Puzar. 2012. TAO: How Facebook Serves the Social Graph. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 791–792. https://doi.org/10.1145/2213836.2213957

[65] Qian Wang, Kui Ren, Minxin Du, Qi Li, and Aziz Mohaisen. 2017. SecGDB: Graph Encryption for Exact Shortest Distance Queries with Efficient Updates. In *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10322)*, Aggelos Kiayias (Ed.). Springer, 79–97. https://doi.org/10.1007/978-3-319-70972-7_5

[66] David J. Wu, Joe Zimmerman, Jérémy Planul, and John C. Mitchell. 2016. Privacy-Preserving Shortest Path Computation. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society. https://doi.org/10.14722/ndss.2016.23052

[67] Junhua Zhang, Wentao Li, Long Yuan, Lu Qin, Ying Zhang, and Lijun Chang. 2022. Shortest-Path Queries on Complex Networks: Experiments, Analyses, and Improvement. *Proc. VLDB Endow.* 15, 11 (jul 2022), 2640–2652. https://doi.org/10.14778/3551793.3551820

```
ComputeHLD($T, v$) ⟶ $|T[v]|, T_A$
    if $v$ is a leaf then
        // The subtree rooted at $v$ is of size 1.
        return $1, T$
    else
        $v\_size = 1$
        $temp = \{\}$
        // Compute size of subtree rooted at $v$.
        for child $w$ of $v$ do
            $w\_size, T \leftarrow$ ComputeHLD($T, w$)
            $temp[w] \leftarrow w\_size$
            $v\_size \leftarrow v\_size + w\_size$
        for $(w, w\_size)$ in $temp$ do
            // Determine if $(w, v)$ is heavy or light.
            if $w\_size < v\_size/2$ then
                Label $(w, v)$ in $T$ as “light”
            else
                Label $(w, v)$ in $T$ as “heavy”
        return $v\_size, T$
```

Figure 8: Psuedocode for ComputeHLD.

A HEAVY-LIGHT DECOMPOSITION

Heavy-light decomposition (HLD) was introduced by Sleator and Tarjan in order to develop fast algorithms for a number of tree problems, including computing nearest common ancestors and various network flow problems [62].

Definition 22. *Given a rooted tree T, the* ***size of a node*** *v* ***in*** *T, $\text{size}_T(v)$, is the number of nodes in the subtree rooted at node v (the size includes the node v itself).*

Definition 23. *Let T be a rooted tree. An edge between a node v in T and its parent is defined as* ***heavy*** *if and only if $\text{size}_T(v) > \frac{1}{2}\text{size}_T(\text{parent}(v))$. All other edges in the tree are* ***light****.*

Note that, by definition, any node in a tree has at most one child linked to by a heavy edge.

Definition 24. *The heavy edges decompose the tree nodes of a tree T into vertex disjoint paths which are called* ***heavy chains****. These paths are connected to each other via light edges.*

Theorem 25. *([62]) Let T be a rooted tree with n nodes. From any node in T, the number of light edges needed to reach the root of the tree is at most $\log n$. Thus, the number of heavy chains along the path from any node to the root is also $O(\log n)$.*

Complexity. This algorithm marks each edge while exploring the input tree T using DFS. The run time is the same time and space as running DFS on a tree i.e. $O(n)$.

B PROOFS

B.1 Proof of Lemma 5

Proof. We must prove two properties: (1) $G|_{P_v}$ is connected and (2) there are no cycles in $G|_{P_v}$. By definition of P_v, all paths must end at v. Thus any two vertices $u, u' \in V_{P_v}$ are connected through an undirected path via v.

We now prove property (2). Let the vertices in V be labeled from 1 to n and let $A(u, v, k)$ denote the shortest path from u to v using only vertices 1 to k. Recall the recurrence used by Floyd-Warshall.

$$A(u, v, k) = \min \begin{cases} A(u, v, k-1) \\ A(u, k, k-1) + A(k, v, k-1) \end{cases}$$

where $A(u, v, 0) = (u, v)$. For a contradiction, suppose that the recurrence produces a cycle in $G|_{P_v}$. Then there must be two distinct paths between a pair of vertices in P_v as depicted in Figure 9.

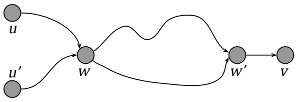


Figure 9: A cycle in P_v.

As k iterates over $[n]$, we update the shortest path from u to v using only vertices $1, \ldots, k$. Similarly for the shortest path from u' to v. However, a cycle as in Figure 9 would imply that for $k = w$ we have $A(u, v, k) = A(u, k, k-1) + A(k, v, k-1)$ and $A(u', v, k) = A(u', k, k-1) + A(k, v, k-1)$. Yet the path corresponding to $A(k, v, k-1)$ in the two equations must have been distinct. This is a contradiction. Thus P_v cannot contain a cycle. □

B.2 Proof of Lemma 26

Lemma 26. *Let λ be an integer and $G = (V, E)$ be a graph on n vertices. Let $K \leftarrow$ PathGES.KeyGen(1^λ) and $(\mathbf{EM}_1, \mathbf{EM}_2) \leftarrow$ PathGES.Encrypt(K, G). Before line 28 (Figure 2) is executed, M_1 contains at most $n^2 \log n$ values.*

Proof. M_1 maps each SDSP query to a token set, each token corresponding to a fragment. By Theorem 25, a path from any vertex u to the root r in an annotated tree T_r^D comprises of no more than $\lfloor \log n \rfloor$ disjoint paths. Since T_r^D is a tree, the path from any vertex v to root r cannot cross a disjoint path more than once. It follows that each query corresponds to at most $\log n$ fragments. There are $n^2 - n$ distinct SPSP queries. Thus, before executing line 28 (Figure 2), M_1 contains at most $n^2 \log n$ values. □

B.3 Proof of Lemma 27

Lemma 27. *Let λ be an integer and $G = (V, E)$ be a graph on n vertices. Let $K \leftarrow$ PathGES.KeyGen(1^λ) and $(\mathbf{EM}_1, \mathbf{EM}_2) \leftarrow$ PathGES.Encrypt(K, G). Before line 28 (Figure 2) is executed, M_2 contains at most $4n^2$ values.*

Proof. On line 15 (Figure 2), each edge-disjoint path of length ℓ is padded to the next power of two. In the worst case $\ell = 2^k + 1$ for some integer k, which results in padding the path up to $2 \cdot 2^k$. The length of the padded path is thus at most $2\ell - 2$. Moreover, storing the canonical fragments of each path requires an additional 2^{k+1}

storage. M_2 maps fragment identifiers to fragments. Before line 28 is executed, M_2 contains at most $4n^2$ edges from the fragments. □

B.4 Proof of Theorem 8

Proof. First we show that the response is the shortest path from u to r. Let T_r^D be the annotated SDSP tree rooted at r. We proceed by induction on the disjoint subpaths as they are discovered using BFS. Let $p_{v_0,r}$ be the first disjoint path, and let $\{p_{v_0,r}^{(j)}\}_{j\in[0,\ell]}$ be its set of canonical fragments. Then for every $j \in [1,\ell]$ and every non-pad vertex w in $p_{v_0,r}^{(j)} \setminus p_{v_0,r}^{(j-1)}$ it is easy to see that $\mathsf{M}_1[(w,r)]$ corresponds to the fragment needed to recover the path from w to r. Suppose that this also holds true for the first $k-1$ subpaths. Let $p_{u,v}$ be the k-th path discovered. By the inductive hypothesis and correctness of BFS, then $\mathsf{M}_1[(u,r)]$ must have been computed correctly. Thus for every $j \in [1,\ell]$ and every non-pad vertex w in $p_{u,v}^{(j)} \setminus p_{u,v}^{(j-1)}$ we have that $\mathsf{M}_1[(w,r)]$ can be computed using the fragments comprising the path from u to r, plus the fragment $p_{u,v}^{(j)}$. Moreover, since these fragments form paths in the SDSP tree T_r, then the path is a shortest path in G.

We now claim that each fragment contains no more than 2 times the edges contained in the true shortest path. Let $p_{u,v}$ be a subpath covering the queried shortest path. Let $p = (w_0, w_1, \dots, w_k, v)$ in $p_{u,v}$ be the vertices precisely contained in the queried shortest path. Let 2^k be the smallest power of two at least as large as the length of p. Then there exists fragment $p_{u,v}^{(k)}$ that covers p and is at most 2 times the length of p. The for loop on line 23 of Encrypt ensures that each query is mapped to the minimum length fragment.

Storage complexity follows from the fact that we pad M_1 and M_2 up to $n^2 \log n$ and $4n^2$ values, respectively. Recall that we take the overhead of the encrypted EMM to be linear in the number of values. Computing the SDSP trees takes time $O(n^3)$ for general graphs. Each tree in $\{T_r\}_{r\in V}$ is traversed once and has a maximum size of $O(n)$. The total set of fragments for a given SDSP tree require no more than 2 times the space of the original tree. For each tree, computing the fragments and the multimaps can be done in time linear in n. Thus the total time to encrypt is $O(n^3)$.

Issuing a query requires computing a single token using the EMM-RR.Token. This takes $O(1)$ time and returns a single token of size linear in λ. Any call to Search results in one look-up to $\mathbf{EM}_1$, $\lfloor \log n \rfloor$ look-ups to $\mathbf{EM}_2$, and the retrieval of at most $2t$ encrypted edges, for a total running time of $O(\log n + t)$. Executing Reveal to decrypt the fragments is linear in the number of edges in the response, and thus takes $O(t)$ time. □

B.5 Proof of Theorem 9

Proof. In Figure 10, we describe a stateful algorithm $\mathcal{S}$ that simulates the view of the adversary in the **Ideal** game. We use the simulators of EMM-RR and EMM-RH to achieve this and denote them as $\mathcal{S}_{\mathsf{RR}}$ and $\mathcal{S}_{\mathsf{RH}}$, respectively. We now use the following hybrid sequence to prove computational indistinguishability between the output of the simulator and PathGES.

Hyb 0: This is equal to the **Real** game.

Hyb 1: Identical to **Hyb 0**, except instead of invoking algorithms EMM-RH.KeyGen and EMM-RH.Setup, invoke $\mathcal{S}_{\mathsf{RH}}$.SimS on input $4n^2$. An adversary cannot distinguish between these two sequential hybrids with more than non-negligible advantage, otherwise it would break the adaptive security of EMM-RH.

```
SimS(1^λ, $\mathcal{L}_S(G)$) → ($\mathbf{ED}$, st)
    ($\mathbf{EM}_1$, st_1) ← $\mathcal{S}_{RR}$.SimS($1^\lambda$, $n^2 \log n$)
    ($\mathbf{EM}_2$, st_2) ← $\mathcal{S}_{RH}$.SimS($1^\lambda$, $4n^2$)
    Initialize empty matrix $A'$
    Initialize empty sequences $B, C$
    Initialize empty tables $T, T'$
    return (($\mathbf{EM}_1$, $\mathbf{EM}_2$), ($A', B, C, T, T'$))

SimQ($1^\lambda$, $\mathcal{L}_Q(G, (q_1, \dots, q_k))$) → (tk, st)
    // Parse leakage as matrix $A$ and graph $H = (I \cup F, E')$
    ($A, H$) ← $\mathcal{L}_Q(G, (q_1, \dots, q_k))$
    // Update state for $k$th query
    ($A', B, C, T, T'$) ← st
    Let $c$ be the neighborhood size of vertex $k \in I$
    Let $m = |E'|$ be the number of edges in $H'$.
    $Tokens$ ← $\emptyset$
    for $i = m - c + 1, \dots, m$ do
        // Derive leakage of EMM-RH.
        Append a column and row to $A'$
        for $j \leq i$ do
            $A'[i,j] = A'[j,i]$
                $= \begin{cases} 1 & \text{if } e_i, e_j \in E' \text{incident} \\ 0 & \text{otherwise.} \end{cases}$
        Append $weight(e_i)$ to $B$
        if $A'[i,j] = 1$ for some $j < i$ then
            // Add previous fragment token.
            tk' ← $T'[j]$
            $T'[i]$ ← tk'
        else
            // Invoke $\mathcal{S}_{RH}$ to generate new fragment token.
            (tk', st_2) ← $\mathcal{S}_{RH}$.SimQ($1^\lambda$, ($A', B$))
            $T'[i]$ ← tk'
        $Tokens$ ← $Tokens \cup \{$tk'$\}$
    Append $Tokens$ to $C$
    if $A[k,j] = 1$ for some $j < k$ then
        // Return token using state.
        $T[k]$ ← $T[j]$
        st ← ($A', B, C, T, T'$)
        return ($T[k]$, st)
    // Else invoke $\mathcal{S}_{RR}$ to generate new token for $k$-th query.
    (tk, st_2) ← $\mathcal{S}_{RR}$.SimQ($1^\lambda$, ($A, C$))
    $T[k]$ ← tk
    st ← ($A', B, C, T, T'$)
    return ($T[k]$, st)
```

Figure 10: Psuedocode for simulator $\mathcal{S}$ = (SimS, SimQ).

Hyb 2: Identical to **Hyb 1**, except instead of invoking algorithms EMM-RR.KeyGen and EMM-RR.Setup, invoke $\mathcal{S}_{\mathsf{RR}}$.SimS on input $n^2 \log n$. An adversary cannot distinguish between these two sequential hybrids with more than non-negligible advantage, otherwise it would break the adaptive security of EMM-RR.

Hyb 3: Identical to **Hyb 2**, except maintain some additional state. First, construct a bipartite graph $H = (I \cup F, E')$ such that $I = [k]$ is the set of query indices and F is the set of identifiers of encrypted fragments returned in response to the k queries. For each $i \in I$ and $f \in F$, if fragment f is in the response to the i-th query, then add edge (i, f) to H and set $weight((j, f))$ to the length of f. Then construct a $k \times k$ binary matrix A such that $A[i, j] = 1$ iff the i-th query corresponds to the same search token as the j-th query. Additionally, initialize an empty matrix A', empty sequences B, C, and empty tables T, T'. Because this hybrid is only storing additional internal state as compared to **Hyb 2**, the distributions of the two hybrids are equal.

Hyb (4, x) for $x \in [k]$: Identical to the previous hybrid, except for the x-th query, invoke the simulator $\mathcal{S}_{\mathsf{RH}}$ using the search and volume pattern of $\mathbf{EM}_2$ derived from graph H. In particular, instead of invoking EMM-RH.Token, update A' and B using H and simulate the tokens used to query $\mathbf{EM}_2$ by invoking $\mathcal{S}_{\mathsf{RH}}$.SimQ as described in Figure 11 .

```
Let $H' \subseteq H$ be the subgraph induced by nodes $[x] \times F$.
Let $c$ be the neighborhood size of vertex $x$ in $H'$.
Let $m$ be the number of edges in $H'$.
$Tokens \leftarrow \emptyset$
for $i = m - c + 1, \ldots, m$ do
    // Derive leakage of EMM-RH.
    Append a column and row to $A'$
    for $j \le i$ do
        $A'[i,j] = A'[j,i] = \begin{cases} 1 & \text{if } e_i, e_j \in E'\text{incident} \\ 0 & \text{otherwise.} \end{cases}$
    Append $weight(e_i)$ to $B$
    if $A'[i,j] = 1$ for some $j < i$ then
        // Add previous fragment token.
        $\mathsf{tk}' \leftarrow T'[j]$
        $T'[i] \leftarrow \mathsf{tk}'$
    else
        // Invoke $\mathcal{S}_{\mathsf{RH}}$ to generate new fragment token.
        $(\mathsf{tk}', \mathsf{st}_2) \leftarrow \mathcal{S}_{\mathsf{RH}}.\mathsf{SimQ}(1^\lambda, (A', B))$
        $T'[i] \leftarrow \mathsf{tk}'$
    $Tokens \leftarrow Tokens \cup \{\mathsf{tk}'\}$
    Append $Tokens$ to $C$
```

Figure 11: Psuedocode for Hyb (4, i).

An adversary cannot distinguish between **Hyb (4, x)** and **Hyb (4, $x - 1$)** (or **Hyb 3** when $x = 1$) with more than non-negligible advantage, otherwise it would break the adaptive security of EMM-RH.

Hyb (5, x) for $x \in [k]$:. Identical to the previous hybrid, except invoke $\mathcal{S}_{\mathsf{RR}}$ for the x-th query. Each query to PathGES results in a look up to $\mathbf{EM}_1$ using the same token used to initiate the SPSP query. Moreover, the response of $\mathbf{EM}_1$ is exactly the set of tokens used to query and retrieve the fragments from $\mathbf{EM}_2$. Let $A[1 : i, 1 : i]$ be a submatrix of A and C be the sequence of token sets computed in the previous hybrids. Instead of invoking EMM-RR.Token – simulate the token using $\mathcal{S}_{\mathsf{RR}}$.SimQ on the leakage $(A[0 : i, 0 : i], C)$ as described in Figure 12.

```
if $A[x, j] = 1$ for some $j < x$ then
    // Return token using state.
    $T[x] \leftarrow T[j]$
    $\mathsf{st} \leftarrow (A', B, C, T, T')$
    return $(T[x], \mathsf{st})$
// Else invoke $\mathcal{S}_{\mathsf{RR}}$ to generate new token for $x$-th query.
$(\mathsf{tk}, \mathsf{st}_2) \leftarrow \mathcal{S}_{\mathsf{RR}}.\mathsf{SimQ}(1^\lambda, (A[1 : x, 1 : x], C))$
$T[x] \leftarrow \mathsf{tk}$
$\mathsf{st} \leftarrow (A', B, C, T, T')$
return $(T[x], \mathsf{st})$
```

Figure 12: Psuedocode for Hyb (5, i).

An adversary cannot distinguish between **Hyb (5, x)** and **Hyb (5, $x - 1$)** (or **Hyb (4, k)** when $x = 1$), otherwise it would break the adaptive security of EMM-RR.

Hyb 6: Identical to **Hyb (5, k)**, except run $\mathcal{S}$.SimQ on the leakage $\mathcal{L}_{\mathsf{Q}}(G, (q_1, \ldots, q_k))$ when a query is issued. We note that given the search pattern A and structure pattern graph H of PathGES, we can run lines 9-39 of Figure 10. Moreover, the A and H obtained from the leakage of PathGES are equal to those computed by the previous hybrid. Thus **Hyb (5, k)** and **Hyb 6** are computationally indistinguishable.

Since the distribution of **Hyb 6** is identical to that of $\mathcal{S}$, then the **Real** and **Ideal** worlds are computationally indistinguishable and the proof follows. □

B.6 Proof of Theorem 16

Proof. Let $(q_1, \ldots, q_k) \in Q$ such that $Q \in \mathcal{Q}$. In Figure 13, we describe a stateful algorithm $\mathcal{T}$ that transforms the leakage of GKT to that of PathGES. This algorithm runs in poly-time and the theorem follows. □

Our proof makes the implicit assumption that the decomposition is known to the transformation algorithm $\mathcal{T}$.

B.7 Proof of Lemma 19

Proof. Let $Q = (q_1, \ldots, q_k)$. We first define π. Let A be the set of queries (a, b) in Q such that $a \in T_v[c]$ and $b = v$. Similarly, let B be the set of queries (a, b) in Q such that $a \in T_w[d]$ and $b = w$.

- If $(a, b) \in A$, then set $\pi((a, b)) = (\phi(a), \phi(b))$.
- If $(a, b) \in B$, then set $\pi((a, b)) = (\phi^{-1}(a), \phi^{-1}(b))$.
- Else set $\pi((a, b)) = (a, b)$.

We now show that π is consistent with the leakage i.e., that $\mathcal{L}_{\mathsf{Q}}(G, \pi(Q)) = \mathcal{L}_{\mathsf{Q}}(G, Q)$. Since π is a permutation and each query in Q is issued once, then $\pi(Q)$ is a sequence of distinct queries. Thus the query pattern for both leakage functions is a $|Q| \times |Q|$ matrix with 1's along the diagonal and 0's everywhere else.

Let $H = (I, F)$ and $H' = (I', F')$ be the structure pattern graphs of $\mathcal{L}_{\mathsf{Q}}(G, Q)$ and $\mathcal{L}_{\mathsf{Q}}(G, \pi(Q))$, respectively. It remains to show that the structure pattern is equal, i.e. that there is an edge-weight preserving isomorphism $\varphi : H \longrightarrow H'$. Observe that the subgraph of H induced by A and its neighbors is disconnected from the rest

```
TSetup($1^\lambda, \mathcal{L}_{\mathsf{S}}^{\mathsf{GKT}}(G)) \rightarrow \mathcal{L}_{\mathsf{S}}^{\mathsf{PathGES}}(G)$
    $n \leftarrow \mathcal{L}_{\mathsf{S}}^{\mathsf{GKT}}(G)$
    return $n$

Tquery($1^\lambda, \mathcal{L}_{\mathsf{Q}}^{\mathsf{GKT}}(G, Q)) \rightarrow \mathcal{L}_{\mathsf{Q}}^{\mathsf{PathGES}}(G, Q)$
    $(\mathsf{QP}, \mathsf{PIP}, \mathbf{t}) \leftarrow \mathcal{L}_{\mathsf{Q}}^{\mathsf{GKT}}(q_1, ..., q_k)$
    Compute the query trees $\{S_i\}_{i \in [n]}$.
    Compute the HLD of each query tree in $\{S_i^D\}$
    Compute fragments of each path in $\{S_i^D\}$.
    Let $I = [k]$ and $F$ be the set of fragments.
    Define graph $H = (I \cup F, E')$ such that $E' = \emptyset$
    for $j = 1, ..., k$: do
        // Add each fragment associated with $q_j$.
        Using PIP find path $p$ in $\{S_i^D\}_{i \in [n]}$ returned.
        Let $F' \subseteq F$ be the set of fragments or minimal length that cover $p$.
        Add $\{(j, f) : f \in F'\}$ to $H$
    Str $\leftarrow H$
    return $(\mathsf{QP}, \mathsf{Str})$
```

Figure 13: Psuedocode for transformer $\mathcal{T} = (\mathsf{TSetup}, \mathsf{Tquery})$. We let $Q = (q_1, \ldots, q_k)$ denote a sequence of queries.

of the graph. Similarly for B. This is because all shortest paths in $T_v[c] \cup (c, v)$ are edge disjoint from all other shortest paths and thus do not share fragments with any other queries.

We now define φ for each of these components of H. Consider the queries $q \in A$. Any edge-disjoint decomposition of T_v must be a decomposition for the subtree $T_v[c]$. Since the rooted-tree isomorphism ϕ is edge-preserving, then any edge-disjoint decomposition of $T_v[c]$ must also be a decomposition for $T_w[d]$ under ϕ. Moreover, ϕ maps shortest paths to shortest paths and fragments of a particular length to fragments of the same length. We abuse notation of $\phi(f)$ to mean the image of the vertices in fragment f under ϕ.

- For edge $(q, f) \in I \times F$ such that $q = (a, b) \in A$, set $\varphi(q) = (\phi(a), \phi(b))$. Since all queries have been issued, $(\phi(a), \phi(b))$ must exist. Also set $\varphi(f) = \phi(f)$.
- For edge $(q, f) \in I \times F$ such that $q = (a, b) \in B$, set $\varphi(q) = (\phi^{-1}(a), \phi^{-1}(b))$. Set $\varphi(f) = \phi^{-1}(f)$.
- For edge $(q, f) \in I \times F$ such that $q \notin A \cup B$, then set φ as the identity, i.e. $\varphi(q) = q$ and $\varphi(f) = f$.

Since φ maps fragments under the identity or ϕ, then it is edge-weight preserving (the edge weights of H and H' correspond to the lengths of the incident fragments). To see that φ is also edge-preserving, let (q, f) be any edge in H where $q = (a, b)$. If $q \in A$ or $q \in B$, then by isomorphism of ϕ either $((\phi(a), \phi(b)), \phi(f)) \in H'$ or $((\phi^{-1}(a), \phi^{-1}(b)), \phi^{-1}(f)) \in H'$, respectively. Else $q \notin (A \cup B)$, in which case $(\varphi(q), \varphi(f)) = (q, f) \in H'$. Thus $H \cong H'$ and π is consistent with $\mathcal{L}_{\mathsf{Q}}(G, Q)$. □

C RECONSTRUCTION SPACE BOUNDS

Theorem 28. *Let $n \geq 5$. Let K_n be the complete graph, L_n the line graph, S_n the asymmetric star, and G_n be the square grid graph. Let Q be any sequence in which each SPSP is issued exactly once. Let K_n, L_n, S_n and G_n all be encrypted using* GKT *[26]. Then we have that following bounds:*

(1) $|\mathsf{QRS}_{\mathsf{GKT}}(K_n, Q)| = (n!)^{n+1}$
(2) $|\mathsf{QRS}_{\mathsf{GKT}}(L_n, Q)| = 2^{n/2}$
(3) $|\mathsf{QRS}_{\mathsf{GKT}}(S_n, Q)| = 1$
(4) $|\mathsf{QRS}_{\mathsf{GKT}}(G_n, Q)| \leq \left(2^{\frac{7n - 2\sqrt{n} + 4}{4}}\right)$

Theorem 29. *Let $n \geq 5$. Let K_n be the complete graph, L_n the line graph, S_n the asymmetric star, and G_n be the square grid graph. Let Q be any sequence in which each SPSP is issued exactly once. Let K_n, L_n, S_n and G_n all be encrypted using* PathGES *(Figure 2). Then we have that following bounds:*

(1) $|\mathsf{QRS}_{\mathsf{PathGES}}(K_n, Q)| = (n^2)!$
(2) $|\mathsf{QRS}_{\mathsf{PathGES}}(L_n, Q)| \geq (n!)^{\log n - 1}$
(3) $|\mathsf{QRS}_{\mathsf{PathGES}}(S_n, Q)| \geq n \prod_{i=0}^{\log(n+1)-3} (2^i!)$
(4) $|\mathsf{QRS}_{\mathsf{PathGES}}(G_n, Q)| \geq \left(\frac{\sqrt{n}}{4}!\right)^{\frac{\sqrt{n}}{2} n}$

C.1 Complete Graph

Lemma 30. *Let $G = (V, E)$ be a complete graph on n vertices and Q be any sequence in which each SPSP is issued exactly once. Let G be encrypted using the GKT scheme. Then $|\mathsf{QRS}_{\mathsf{GKT}}(G, Q)| = (n!)^{n+1}$.*

Proof. Given the query leakage, one can build n complete SDSP trees [23]. The SDSP trees of a complete graph are n stars, each rooted at a different node of G. Each star comprises n nodes. There are also n query trees, all of which are isomorphic stars. There are $n!$ ways to match the query trees and SDSP trees. For each query-SDSP tree pair, there are $n!$ isomorphisms between the two trees. Since there are n such pairs, the total size of the query reconstruction space is $n! \cdot (n!)^n$. □

Lemma 31. *Let $G = (V, E)$ be a complete graph on n vertices and Q be any sequence in which each SPSP is issued exactly once. Let G be encrypted using* PathGES *(Figure 2). Then $|\mathsf{QRS}_{\mathsf{PathGES}}(G, Q)| = (n^2)!$.*

Proof. The SDSP trees are n stars. Each tree results in n edge-disjoint paths each of length 1. In total there are n^2 fragments all of which are indistinguishable. There are $(n^2)!$ bijections between $V \times V$ and the set of n^2 fragments and the lemma follows. □

C.2 Line Graph

Lemma 32. *Let $G = (V, E)$ be a line graph on n vertices and Q be any sequence in which each SPSP is issued exactly once. Let G be encrypted using the GKT scheme. Then $|\mathsf{QRS}_{\mathsf{GKT}}(G, Q)| = 2^{n/2}$.*

Proof. Denote the path as $(1, 2, 3 \ldots, n)$ and let n be even; a similar argument holds for odd n. Given the query leakage, one can build n complete SDSP trees [23]. The SDSP tree rooted at node $i \in V$ is isomorphic to the SDSP tree rooted at $n - i + 1$. Given the set of n query trees, each query tree is isomorphic to exactly two

SDSP trees. There exists only one isomorphism from any query tree to an SDSP tree. This is because each tree has at most two paths of differing lengths descending from the root, so the isomorphism is determined. The reconstruction space is the number of ways to pair the SDSP trees with isomorphic query trees multiplied by the number of isomorphisms between each pair, i.e. $2^{n/2}$. □

Lemma 33. *Let $G = (V, E)$ be a line graph on n vertices and Q be any sequence in which each SPSP is issued exactly once. Let G be encrypted using* PathGES *(Figure 2). Then* $|\mathrm{QRS}_{\mathrm{PathGES}}(G, Q)| = \Omega((n!)^{\log n-1})$.

Proof. Let G be the path $(1, 2, 3 \ldots, n)$. Each SDSP tree comprises at most two paths descending from the root. When the SDSP tree is decomposed, each path becomes disjoint. Any query results in a response comprised of a single canonical fragment. Thus, each canonical fragment of the same length and which does not comprise of any padding vertices are indistinguishable. To obtain this lower bound we will count the number of canonical fragments of the same length. There are at least n canonical fragments of each length $1, 2, 2^2, \ldots, 2^{\log n-1}$. Each canonical fragment of length ℓ length is indistinguishable from any other fragment of the same length. This gives us a lowerbound of $(n!)^{\log n-1}$. □

C.3 Asymmetric Star

We define the **asymmetric star** $S_{k_1,\ldots,k_\ell}$ as the graph with one central vertex and ℓ paths of distinct lengths $k_1 \ldots, k_\ell$ which are incident to the central vertex. For convenience, we consider the asymmetric star $S_{2^1,\ldots,2^N}$ for integer N.

Lemma 34 ([23]). *Let $G = (V, E)$ be an asymmetric star $S_{2^1,\ldots,2^N}$ on $n = 2^{N+1} - 1$ vertices and Q be any sequence in which each SPSP is issued exactly once. Let G be encrypted using the GKT scheme. Then* $|\mathrm{QRS}_{\mathrm{GKT}}(G, Q)| = 1$.

More generally, under the leakage of the GKT scheme, all asymmetric stars are fully query recoverable.

Lemma 35. *Let $G = (V, E)$ be the asymmetric star $S_{2^1,\ldots,2^N}$ on $n = 2^{N+1} - 1$ vertices and Q be any sequence in which each SPSP is issued exactly once. Let G be encrypted using* PathGES *(Figure 2). Then* $|\mathrm{QRS}_{\mathrm{PathGES}}(G, Q)| = \Omega(n \prod_{i=0}^{\log(n+1)-3}(2^i!))$.

Proof. Let $j \in [N]$ and consider an SDSP tree rooted at a vertex along the path of length 2^j. This tree consists of the other paths of lengths $\{2^i\}_{j \neq i, i \in [N]}$ all descending from the fixed root. Each path of length 2^i gives rise to a fragment of length at least 2^i. Applying Corollary 21, each such fragment results in 2^{i-1} indistinguishable queries. Because these queries are indistinguishable, the assignment can permute these in any way possible i.e. there exist at least $\prod_{i=1, i\neq j}^{N}(2^{i-1}!)$ distinct assignments. Thus each SDSP tree results in $\prod_{i=1, i\neq j}^{N}(2^{i-1}!) \geq \prod_{i=0}^{N-2}(2^i!)$ possible assignments consistent with the leakage. There are n SDSP trees (one rooted at each vertex in G). This yields a lower bound of $n \prod_{i=0}^{N-2}(2^i!) = n \prod_{i=0}^{\log(n+1)-3}(2^i!)$ on the size of the query reconstruction space. □

C.4 Grid Graph

In this section, we explore the reconstruction space of an $n = N \times N$ grid graph, which we denote as $G_{N\times N}$. This graph is insightful as it approximates the geographic graph of a grid-like city such as New York. Note that for any given graph $G_{N\times N} = (V, E)$ and any vertex $v \in V$, there are a number of possible SDSP trees and we thus we make an additional assumption about how the SDSP trees are computed. In particular, for any node $v \in V$, the SDSP tree is computed by drawing one vertical path the length of the grid and then drawing horizon paths branching off the vertical path in either direction (See Figure 14). We will prove our theorems with respect to odd N as more symmetries arise in grids of odd length.

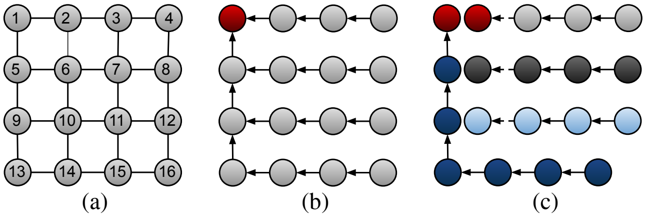


Figure 14: (a) A 4×4 grid graph, (b) its SDSP graph rooted at vertex 1, and (c) its corresponding edge-disjoint decomposition.

Lemma 36. *Let $G = (V, E)$ be a grid graph on $n = N \times N$ vertices and Q be any sequence in which each SPSP is issued exactly once. Let G be encrypted using* GKT *with SDSP trees computed as described above. Then* $|\mathrm{QRS}_{\mathrm{GKT}}(G, Q)| \leq \left(2^{\frac{7n-2\sqrt{n}+4}{4}}\right)$.

Proof. Let $N \geq 5$ be odd; a similar argument holds for even N. For convenience, assume a fixed orientation of G such that its edges are oriented either horizontally or vertically. Given the query leakage, one can build n complete SDSP trees [23]. Note that any vertex $v \in V$ falls in one of the following 5 categories: (1) v is the center node, (2) v lies along the central vertical axis but is not the center node, (3) v lies along the central horizontal axis but is not the center node, (4) v is along a diagonal but is not the center node, or (5) none of the above.

Case (1): there is no $w \in V \setminus \{v\}$ such that $T_v \cong T_w$. Given T_v, one can reflect each of the N horizontal paths independently of each other, giving rise to 2^N isomorphisms between the query tree S_v and the SDSP tree T_v. Rotating S_v 180° also results in another 2^N possible isomorphisms.

Case (2): there are $N - 1$ such nodes. For each such v there exists 1 node $w \in V \setminus \{v\}$ such that $T_v \cong T_w$. There are 2 ways to match the query trees and SDSP trees. As before, each horizontal path can be independently reflected of the others giving rise to 2^N isomorphisms between each query-plaintext tree pair.

Case (3): there are $N - 1$ such nodes. For such v there exists 1 node $w \in V \setminus \{v\}$ such that $T_v \cong T_w$. There are 2 ways to pair the query trees and the SDSP trees, and 2 isomorphisms between each pair of trees obtained by reflecting across the horizontal axis.

Case (4): there exist $2(N - 1)$ such nodes. For each node $w \in V \setminus \{v\}$ there exist 3 other nodes $w \in V \setminus \{v\}$ such that $T_v \cong T_w$. There are 4! ways to pair the query trees with the SPSP trees. For each pair, the isomorphism is unique.

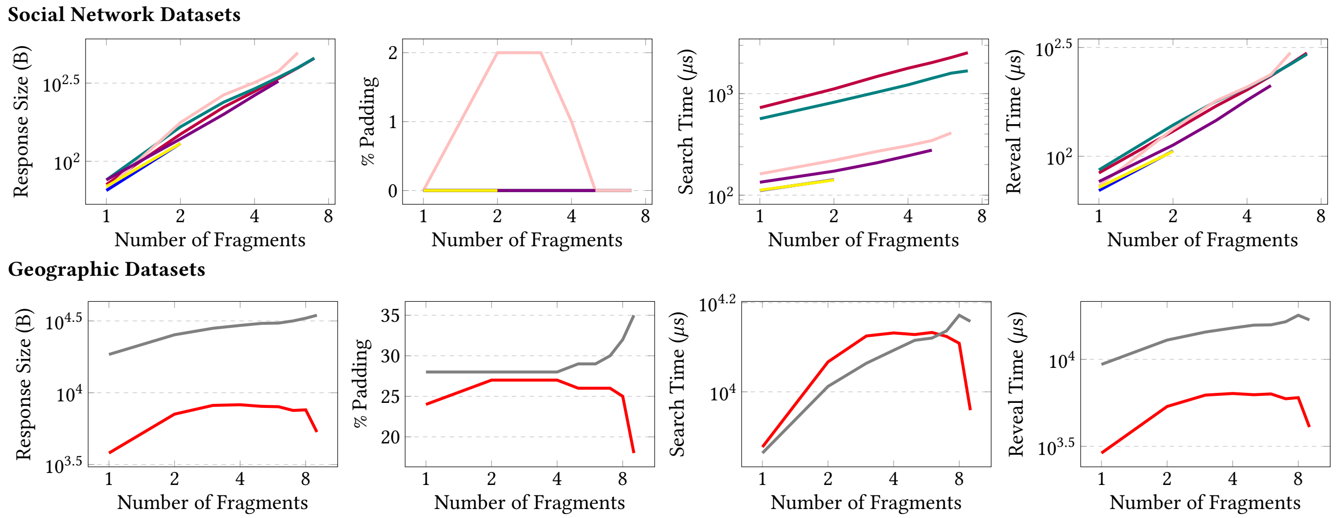


Figure 15: Query benchmarks with respect to the number of fragments. We use the following symbols for the social network datasets: InternetRouting (▬), Ca-GrQc (▬), email-EU-core (▬), facebook (▬), p2p-Gnutella08 (▬), p2p-Gnutella04 (▬). And the following symbols for the geographic datasets: Swiss (▬) and Cali (▬). For each dataset we issued 100,000 random queries, partitioned them based on number of fragments, and took the average of the respective attribute of each set in the partition.

Case (5): there exist $N^2 - 1 - (4n - 4)$ such nodes. For each v there exist 3 other $w \in V \setminus \{v\}$ such that $T_v \cong T_w$. There are 4! ways to match the query trees and SDSP trees, and for each pair the isomorphism is unique.

Lastly we note that reflecting the grid across either diagonal gives us an additional 4 different configurations in total. Putting all this together yields a query reconstruction space of size:

$$4\left(2^{N+1}\cdot\left(2^{N+1}\right)^{\frac{N-1}{2}}\cdot 4^{\frac{N-1}{2}}\cdot(4!)^{\frac{N-1}{2}}\cdot(4!)^{\frac{N^2-1-(4N-4)}{4}}\right)$$
$$\leq\left(2^{\frac{7n-2\sqrt{n}+4}{4}}\right)$$

□

Lemma 37. *Let $G = (V, E)$ be a grid graph on $n = N \times N$ vertices and Q be any sequence in which each SPSP is issued exactly once. Let G be encrypted using* PathGES *(Figure 2) with SDSP trees computed as described above. Then*

$$|\mathrm{QRS}_{\mathsf{PathGES}}(G, Q)| \geq \left(\frac{\sqrt{n}}{4}!\right)^{\frac{\sqrt{n}}{2}n} \tag{1}$$

Proof. Let $v \in V$ and T_v be the SDSP tree rooted at v; this tree can be divided into 4 quadrants with v at the center. The largest of these quadrants is at least $\frac{N}{2} \times \frac{N}{2}$ vertices. For any v, the HLD decomposition induces the horizontal paths in the largest quadrant to all be disjoint. Each of these disjoint paths is of length $\frac{N}{2}$ and results in $\log\frac{N}{2}$ fragments of lengths $1, 2, 2^2 \ldots, 2^{\log\frac{N}{2}}$.

Each fragment of length 2^k for some k results in at least 2^{k-1} indistinguishable queries. An assignment consistent with the leakage can permute the queries associated with a fragment of length 2^{k-1} in $(2^{k-1})!$. Note that it must be a permutation since we assumed that each query was issued exactly once. A single path of length $\frac{N}{2}$ thus results in at least $\prod_{i=1}^{\log\frac{N}{2}}(2^{i-1})!$ assignments.

There are at least $\frac{N}{2}$ such paths in any SDSP tree and n possible SDSP trees. Putting this together and replacing N with $\sqrt{n}$ yields a lower bound of

$$\left(\prod_{i=1}^{\log\frac{\sqrt{n}}{2}}\left(2^{i-1}\right)!\right)^{\frac{\sqrt{n}}{2}n} \geq \left(\frac{\sqrt{n}}{4}!\right)^{\frac{\sqrt{n}}{2}n}$$

□

D ADDITIONAL EMPIRICAL RESULTS

In Figure 15, we depict the query benchmarks for all datasets, partitioned by the number of fragments. We find that, in general, the query benchmarks of the social network datasets vary linearly with the number of fragments. In contrast, for the geographic datasets, the query benchmarks level off for responses comprised of more than one fragment. This is likely because the paths in the social network graphs consist of small fragments and path lengths increase relative to the number of fragments, whereas the fragment lengths of the geographic graphs are much longer. Queries on the geographic graphs comprised of few fragments may still correspond to long paths, resulting in larger response sizes, more padding, and increased query time.