GRECS: Graph Encryption for Approximate Shortest Distance Queries

Xianrui Meng[*]
xmeng@cs.bu.edu

Seny Kamara[†]
senyk@microsoft.com

Kobbi Nissim[‡]
kobbi@cs.bgu.ac.il

George Kollios[§]
gkollios@cs.bu.edu

Abstract

We propose graph encryption schemes that efficiently support approximate shortest distance queries on large-scale encrypted graphs. Shortest distance queries are one of the most fundamental graph operations and have a wide range of applications. Using such graph encryption schemes, a client can outsource large-scale privacy-sensitive graphs to an untrusted server without losing the ability to query it. Other applications include encrypted graph databases and controlled disclosure systems. We propose **GRECS** (stands for **GR**aph **E**n**C**ryption for approximate **S**hortest distance queries) which includes three schemes that are provably secure against any semi-honest server. Our first construction makes use of only symmetric-key operations, resulting in a computationally-efficient construction. Our second scheme, makes use of somewhat-homomorphic encryption and is less computationally-efficient but achieves optimal communication complexity (i.e., uses a minimal amount of bandwidth). Finally, our third scheme is both computationally-efficient and achieves optimal communication complexity at the cost of a small amount of additional leakage. We implemented and evaluated the efficiency of our constructions experimentally. The experiments demonstrate that our schemes are efficient and can be applied to graphs that scale up to 1.6 million nodes and 11 million edges.

1 Introduction

Graph databases that store, manage, and query large graphs have received increased interest recently due to many large-scale database applications that can be modeled as graph problems. Example applications include storing and querying large Web graphs, online social networks, biological networks, RDF datasets, and communication networks. As a result, a number of systems have been proposed to manage, query, and analyze massive graphs both in academia (e.g., Pregel [MAB+10], GraphLab [LGK+10], Horton [SEHK12], Trinity [SWL13], TurboGraph [HLP+13], and GraphChi-DB [KG14]) and industry (e.g., Neo4j, Titan, DEX, and GraphBase). Furthermore, with the advent of cloud computing, there is a natural desire for enterprises and startups to outsource the storage and management of their databases to a cloud provider. Increasing concerns about data security

[*]Department of Computer Science, Boston University. Supported by by NSF grants IIS-1320542 and CNS-1414119.

[†]Microsoft Research.

[‡]Department of Computer Science, Ben-Gurion University and CRCS, Harvard University. Work partly done when the author was visiting the Rafik B. Hariri Institute for Computing and Computational Science & Engineering at Boston University. Supported by NSF grant CNS-1237235, a Simons Investigator grant, and ISF grant 276/12.

[§]Department of Computer Science, Boston University. Supported by by NSF grants IIS-1320542 and CNS-1414119.

and privacy in the cloud, however, have curbed many data owners' enthusiasm about storing their databases in the cloud.

To address this, Chase and Kamara [CK10] introduced the notion of graph encryption. Roughly speaking, a graph encryption scheme encrypts a graph in such a way that it can be privately queried. Using such a scheme, an organization can safely outsource its encrypted graph to an untrusted cloud provider without losing the ability to query it. Several constructions were described in [CK10] including schemes that support adjacency queries (i.e., given two nodes, do they have an edge in common?), neighbor queries (i.e., given a node, return all its neighbors) and focused subgraph queries on web graphs (a complex query used to do ranked web searches). Graph encryption is a special case of *structured encryption*, which are schemes that encrypt data structures in such a way that they can be privately queried. The most well-studied class of structured encryption schemes are *searchable symmetric encryption* (SSE) schemes [SWP00, CM05, Goh03, CGKO06, KPR12, KP13, CJJ$^+$13, CJJ$^+$14, NPG14, SPS14] which, roughly speaking, encrypt search structures (e.g., indexes or search trees) for the purpose of efficiently searching on encrypted data.

In this work, we focus on the problem of designing graph encryption schemes that support one of the most fundamental and important graph operations: finding the shortest distance between two nodes. Shortest distance queries are a basic operation in many graph algorithms but also have applications of their own. For instance, on a social network, shortest distance queries return the shortest number of introductions necessary for one person to meet another. In protein-protein interaction networks they can be used to find the functional correlations among proteins [PWJ04] and on a phone call graph (i.e., a graph that consists of phone numbers as vertices and calls as edges) they return the shortest number of calls connecting two nodes.

Our techniques and contributions. Computing shortest distance queries on massive graphs (e.g., the Web graph, online social networks or a country's call graph) can be very expensive, so in practice one typically pre-computes a data structure from the graph called a *distance oracle* that answers shortest distance queries *approximately* [TZ05, DSGNP10, CDF$^+$13]; that is, given two vertices v_1 and v_2, the structure returns a distance d that is at most $\alpha \cdot \mathsf{dist}(v_1, v_2) + \beta$, where $\alpha, \beta > 1$ and $\mathsf{dist}(v_1, v_2)$ is the exact distance between v_1 and v_2.

In this work, we focus on designing structured encryption schemes for a certain class of distance oracles referred to as *sketch-based* oracles. Below we summarize our contributions:

- We propose three distance oracle encryption schemes. Our first scheme only makes use of symmetric-key operations and, as such, is very computationally-efficient. Our second scheme makes use of somewhat-homomorphic encryption and achieves optimal communication complexity. Our third scheme is computationally-efficient, achieves optimal communication complexity and produces compact encrypted oracles at the cost of some leakage.

- We show that all our constructions are adaptively semantically-secure with reasonable leakage functions.

- We implement and evaluate our solutions on real large-scale graphs and show that our constructions are practical.

1.1 Related Work

Graph privacy. Privacy-preserving graph processing has been considered in the past. Most of the work in this area, however, focuses on privacy models that are different than ours. Some of the proposed approaches include structural anonymization to protect neighborhood information

[GYJ$^+$11, LT08, CFL10], use differential privacy [DMNS06] to query graph statistics privately [KNRS13, SY13], or use private information retrieval (PIR) [MY12] to privately recover shortest paths. We note that none of these approaches are appropriate in our context where the graph itself stores sensitive information (and therefore must be hidden unlike in the PIR scenario) and is stored remotely (unlike the differential privacy and anonymization scenarios). Structured and graph encryption was introduced by Chase and Kamara in [CK10]. Structured encryption is a generalization of searchable symmetric encryption (SSE) which was first proposed by Song, Wagner and Perrig [SWP00]. The notion of adaptive semantic security was introduced by Curtmola, Garay, Kamara and Ostrovsky in [CGKO06] and generalized to the setting of structured encryption in [CK10]. One could also encrypt and outsource the graph using fully homomorphic encryption [Gen09], which supports arbitrary computations on encrypted data, but this would be prohibitively slow in practice. Another approach is to execute graph algorithms over encrypted and outsourced graphs is to use Oblivious RAM [GO96] over the adjacency matrix of the graph. This approach, however, is inefficient and not practical even for small graphs since it requires storage that is quadratic in the number of nodes in the graph and a large number of costly oblivious operations. Recent work by [WNL$^+$14] presents an oblivious data structure for computing shortest paths on planar graphs using ORAM. For a sparse planar graph with $O(n)$ edges, their approach requires $O(n^{1.5})$ space complexity at the cost of $O(\sqrt{n}\log n)$ online query time. Recent works based on ORAM, such as [**?**, **?**], also propose oblivious secure computation frameworks that can be used to compute single source shortest paths. However, these are general purpose frameworks and are not optimized to answer shortest distance queries. Other techniques, such as those developed by Blanton, Steele and Aliasgari [BSA13] and by Aly et al. [ACM$^+$13] do not seem to scale to sparse graphs with millions of nodes due to the quadratic complexity of the underlying operations which are instantiated with secure multi-party computation protocols.

Distance oracles. Computing shortest distances on large graphs using Dijkstra's algorithm or breadth first search is very expensive. Alternatively, it is not practical to store all-pairs-shortest-distances since it requires quadratic space. To address this, in practice, one pre-computes a data structure called a *distance oracle* that supports *approximate* shortest distance queries between two nodes with logarithmic query time. Solutions such as [DSGNP10, PBCG09, QXSW13, CDF$^+$13, Che14, Coh14, CHKZ03] carefully select seed nodes (also known as landmarks) and store the shortest distances from all the nodes to the seeds. The advantage of using such a data structure is that they are compact and the query time is very fast. For example, the distance oracle construction of Das Sarma, Gollapudi, Najork and Panigrahy [DSGNP10] requires $\tilde{O}(n^{1/c})$ work to return a $(2c-1)$-approximation of the shortest distance for some constant c.

2 Preliminaries and Notations

Given an undirected graph $G = (V, E)$, we denote its total number of nodes as $n = |V|$ and its number of edges as $m = |E|$. A shortest distance query $q = (u, v)$ asks for the length of the shortest path between u and v which we denote $\mathsf{dist}(u, v)$. The notation $[n]$ represents the set of integers $\{1, \ldots, n\}$. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ. We write $x \xleftarrow{\$} X$ to represent an element x being uniformly sampled at random from a set X. The output x of a probabilistic algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$ and that of a deterministic algorithm $\mathcal{B}$ by $x := \mathcal{B}$. Given a sequence of elements $\mathbf{v}$, we define its i^{th} element either as v_i or $\mathbf{v}[i]$ and its total number of elements as $|\mathbf{v}|$. If A is a set then $|A|$ refers to its cardinality. Throughout, $k \in \mathbb{N}$ will denote the security parameter and we assume all algorithms take k implicitly as input.

A function $\nu : \mathbb{N} \to \mathbb{N}$ is negligible in k if for every positive polynomial $p(\cdot)$ and all sufficiently large $k, \nu(k) < 1/p(k)$. We write $f(k) = \mathsf{poly}(k)$ to mean that there exists a polynomial $p(\cdot)$ such that $f(k) \leq p(k)$ for all sufficiently large $k \in \mathbb{N}$; and we similarly write $f(k) = \mathsf{negl}(k)$ to mean that there exists a negligible function $\nu(\cdot)$ such that $f(k) \leq \nu(k)$ for all sufficiently large k. A dictionary DX is a data structure that stores label/value pairs $(\ell_i, v_i)_{i=1}^n$. Dictionaries support insert and lookup operations defined as follows: an insert operation takes as input a dictionary DX and a label/value pair (ℓ, v) and adds the pair to DX. We denote this as $\mathsf{DX}[\ell] := v$. A lookup operation takes as input a dictionary DX a label ℓ_i and returns the associated value v_i. We denote this as $v_i := \mathsf{DX}[\ell_i]$. Dictionaries can be instantiated using hash tables and various kinds of search trees.

2.1 Cryptographic Tools

Encryption. In this work, we make use of several kinds of encryption schemes including standard symmetric-key encryption and homomorphic encryption. A symmetric-key encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a set of three polynomial-time algorithms that work as follows. Gen is a probabilistic algorithm that takes a security parameter k as input and returns a secret key K; Enc is a probabilistic algorithm that takes as input a key K and a message m and returns a ciphertext c; Dec is a deterministic algorithm that takes as input a key K and a ciphertext c and returns m if K was the key under which c was produced. A public-key encryption scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is similarly defined except that Gen outputs a public/private key pair $(\mathsf{pk}, \mathsf{sk})$ and Enc encrypts messages with the public key pk. Informally, an encryption scheme is CPA-secure (*Chosen-Plaintext-Attack-secure*) if the ciphertexts it outputs do not reveal any partial information about the messages even to an adversary that can adaptively query an encryption oracle. We refer the reader to [KL08] for formal definitions of symmetric-key encryption and CPA-security.

A public-key encryption scheme is homomorphic if, in addition to $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, it also includes an evaluation algorithm Eval that takes as input a function f and a set of ciphertexts $c_1 \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_1)$ through $c_n \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_n)$ and returns a ciphertext c such that $\mathsf{Dec}_{\mathsf{sk}}(c) = f(m_1, \ldots, m_n)$. If a homomorphic encryption scheme supports the evaluation of any polynomial-time function, then it is a fully-homomorphic encryption (FHE) scheme [RAD78, Gen09] otherwise it is a somewhat homomorphic encryption (SWHE) scheme. In this work, we make use of only "low degree" homomorphic encryption; namely, we only require the evaluation of quadratic polynomials. In particular, we need the evaluation algorithm to support any number of additions: $\mathsf{Enc}_{\mathsf{pk}}(m_1 + m_2) = \mathsf{Eval}\big(+, \mathsf{Enc}_{\mathsf{pk}}(m_1), \mathsf{Enc}_{\mathsf{pk}}(m_2)\big)$; and a *single* multiplication: $\mathsf{Enc}_{\mathsf{pk}}(m_1 m_2) = \mathsf{Eval}(\times, \mathsf{Enc}_{\mathsf{pk}}(m_1), \mathsf{Enc}_{\mathsf{pk}}(m_2))$, that is, a ciphertext that results from a homomorphic multiplication cannot be used in another homomorphic multiplication. Concrete instantiations of such schemes include the scheme of Boneh, Goh and Nissim (BGN) [BGN05] based on bilinear maps and the scheme of Gentry, Halevi and Vaikuntanathan [GHV10] based on lattices.

Pseudo-random functions. A pseudo-random function (PRF) from domain $\mathcal{D}$ to co-domain $\mathcal{R}$ is a function family that is computationally indistinguishable from a random function. In other words, no computationally-bounded adversary can distinguish between oracle access to a function that is chosen uniformly at random in the family and oracle access to a function chosen uniformly at random from the space of all functions from $\mathcal{D}$ to $\mathcal{R}$. A pseudo-random permutation (PRP) is a pseudo-random family of permutations over $\mathcal{D}$. We refer the reader to [KL08] for formal definitions of PRFs and PRPs.

3 Distance Oracles

At a high-level, our approach to designing graph encryption schemes for shortest distance queries consists of encrypting a distance oracle in such a way that it can be queried privately. A distance oracle is a data structure that supports approximate shortest distance queries. A trivial construction consists of pre-computing and storing all the pairwise shortest distances between nodes in the graph. The query complexity of such a solution is $O(1)$ but the storage complexity is $O(n^2)$ which is not practical for large graphs.

We consider two practical distance oracle constructions. Both solutions are sketch-based which means that they assign a sketch Sk_v to each node $v \in V$ in such a way that the approximate distance between two nodes u and v can be efficiently (sublinear) computed from the sketches Sk_u and Sk_v. The first construction is by Das Sarma *et al.* [DSGNP10] which is itself based on a construction of Thorup and Zwick [TZ05] and the second is by Cohen *et al.* [CDF+13]. The two solutions produce sketches of the same form and distance queries are answered using the same operation.

Sketch-based oracles. More formally, a sketch-based distance oracle $\mathsf{DO} = (\mathsf{Setup}, \mathsf{Query})$ is a pair of efficient algorithms that work as follows. Setup takes as input a graph G, an approximation factor α and an error bound ε and outputs an oracle $\Omega_G = \{\mathsf{Sk}_v\}_{v \in V}$. Query takes as input an oracle Ω_G and a shortest distance query $q = (u, v)$. We say that DO is (α, ε)-correct if for all graphs G and all queries $q = (u, v)$, $\Pr[\,\mathsf{dist}(u, v) \leq d \leq \alpha \cdot \mathsf{dist}(u, v)\,] \geq 1 - \varepsilon$, where $d := \mathsf{Query}(\Omega_G, u, v)$. The probability is over the randomness of algorithm Setup.

The Das Sarma *et al.* oracle. The Setup algorithm makes $\sigma = \widetilde{\Theta}(n^{2/(\alpha+1)})$ calls to a Sketch sub-routine with the graph G. Throughout, we refer to σ as the oracle's *sampling parameter* and we note that it affects the size of the sketches. During the ith call, the Sketch routine generates and returns a collection of sketches $(\mathsf{Sk}^i_{v_1}, \ldots, \mathsf{Sk}^i_{v_n})$, one for every node $v_j \in V$. Each sketch $\mathsf{Sk}^i_{v_j}$ is a set constructed as follows. During the ith call to Sketch, it samples uniformly at random $\lambda = \log n$ sets of nodes $S_0, \ldots, S_{\lambda-1}$ of progressively larger sizes. In particular, for all $0 \leq z \leq \lambda - 1$, set S_z is of size 2^z. $\mathsf{Sk}^i_{v_j}$ then consists of λ pairs $\{(w_z, \delta_z)\}_{0 \leq z \leq \lambda-1}$ such that w_z is the closest node to v_j among the nodes in S_z and $\delta_z = \mathsf{dist}(v_j, w_z)$. Having computed σ collections of sketches $(\mathsf{Sk}^i_{v_1}, \ldots, \mathsf{Sk}^i_{v_n})_{i \in [\sigma]}$, Setup then generates, for each node $v_j \in V$, a final sketch $\mathsf{Sk}_{v_j} = \bigcup_{i=1}^{\sigma} \mathsf{Sk}^i_{v_j}$. Finally, it outputs a distance oracle $\Omega_G = (\mathsf{Sk}_{v_1}, \ldots, \mathsf{Sk}_{v_n})$. Throughout, we refer to the nodes stored in the node/distance pairs of the sketches as *seeds*.

The Cohen *et al.* oracle. The Setup algorithm assigns to each node $v \in V$ a sketch Sk_v that includes pairs (w, δ) chosen as follows. It first chooses a random rank function $\mathsf{rk} : V \to [0, 1]$; that is, a function that assigns to each $v \in V$ a value distributed uniformly at random from $[0, 1]$. Let $N_d(v)$ be the set of nodes within distance $d - 1$ of v and let $\rho = \Theta(n^{2/(\alpha+1)})$. Throughout, we refer to ρ as the oracle's *rank parameter* and note that it affects the size of the sketches. For each node $v \in V$, the sketch Sk_v includes pairs (w, δ) such that $\mathsf{rk}(w)$ is less than the ρ^{th} value in the sorted set $\{\mathsf{rk}(y) : y \in N_{\mathsf{dist}(u,v)}(v)\}$. Finally it outputs a distance oracle $\Omega_G = (\mathsf{Sk}_{v_1}, \ldots, \mathsf{Sk}_{v_n})$. Like above, we refer to the nodes stored in the node/distance pairs of the sketches as seeds.

Shortest distance queries. The two oracle constructions share the same Query algorithm which works as follows. Given a query $q = (u, v)$, it finds the set of nodes $\mathbf{I}$ in common between Sk_u and Sk_v and returns the minimum over $s \in \mathbf{I}$ of $\mathsf{dist}(u, s) + \mathsf{dist}(s, v)$. If there are no nodes in common, then it returns $\perp$.

$\mathsf{Sk}(u)$: $\{(a,3), (b,3), (e,6), (g,3), (h,4)\}$
$\mathsf{Sk}(v)$: $\{(b,2), (d,1), (e,3), (h,3), (f,7)\}$

Figure 1: Two sketches for nodes u and v. The approximate shortest distance $d = 5$.

4 Distance Oracle Encryption

In this section we present the syntax and security definition for our oracle encryption schemes. There are many variants of structured encryption, including interactive and non-interactive, response-revealing and response-hiding. We consider interactive and response-hiding schemes which denote the fact that the scheme's query operation requires at least two messages (one from client and a response from server) and that queries output no information to the server.

Definition 4.1 (Oracle Encryption) *A distance oracle encryption scheme* $\mathsf{Graph} = (\mathsf{Setup}, \mathsf{distQuery})$ *consists of a polynomial-time algorithm and a polynomial-time two-party protocol that work as follows:*

- $(K, \mathsf{EO}) \leftarrow \mathsf{Setup}(1^k, \Omega, \alpha, \varepsilon)$*: is a probabilistic algorithm that takes as input a security parameter k, a distance oracle Ω, an approximation factor α, and an error parameter ε. It outputs a secret key K and an encrypted graph* EO*.*
- $(d, \perp) \leftarrow \mathsf{distQuery}_{C,S}\big((K, q), \mathsf{EO}\big)$*: is a two-party protocol between a client C that holds a key K and a shortest distance query $q = (u, v) \in V^2$ and a server S that holds an encrypted graph* EO*. After executing the protocol, the client receives a distance $d \geq 0$ and the server receives $\perp$. We sometimes omit the subscripts C and S when the parties are clear from the context.*

For $\alpha \geq 1$ and $\varepsilon < 1$, we say that Graph *is (α, ε)-correct if for all $k \in \mathbb{N}$, for all Ω and for all $q = (u, v) \in V^2$,*

$$\Pr\left[\, d \leq \alpha \cdot \mathsf{dist}(u, v) \,\right] \geq 1 - \varepsilon,$$

where the probability is over the randomness in computing $(K, \mathsf{EO}) \leftarrow \mathsf{Setup}(1^k, \Omega, \alpha, \varepsilon)$ and then $(d, \perp) \leftarrow \mathsf{distQuery}\big((K, q), \mathsf{EO}\big)$.

4.1 Security

At a high level, the security guarantee we require from an oracle encryption scheme is that: (1) given an encrypted oracle, no adversary can learn any information about the underlying oracle; and (2) given the view of a polynomial number of $\mathsf{distQuery}$ executions for an adaptively generated sequence of queries $\mathbf{q} = (q_1, \ldots, q_n)$, no adversary can learn any partial information about either Ω_G or $\mathbf{q}$.

Such a security notion can be difficult to achieve efficiently, so often one allows for some form of leakage. Following [CGKO06, CK10], this is usually formalized by parameterizing the security definition with leakage functions for each operation of the scheme which in this case include the Setup algorithm and $\mathsf{distQuery}$ protocol.

We adapt the notion of adaptive semantic security from [CGKO06, CK10] to our setting to the case of distance oracle encryption.

Definition 4.2 *Let* $\mathsf{Graph} = (\mathsf{Setup}, \mathsf{distQuery})$ *be an oracle encryption scheme and consider the following probabilistic experiments where $\mathcal{A}$ is a semi-honest adversary, $\mathcal{C}$ is a challenger, $\mathcal{S}$ is a* *simulator and $\mathcal{L}_{\mathsf{Setup}}$ and $\mathcal{L}_{\mathsf{Query}}$ are (stateful) leakage functions:*

$\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(1^k)$**:**

- *$\mathcal{A}$ outputs an oracle Ω, its approximation factor α and its error parameter ε.*
- *Given $\mathcal{L}_{\mathsf{Setup}}(\Omega)$, 1^k, α and ε, $\mathcal{S}$ generates and sends an encrypted graph* EO *to $\mathcal{A}$.*
- *$\mathcal{A}$ generates a polynomial number of adaptively chosen queries $(q_1, \ldots, q_m)$. For each q_i, $\mathcal{S}$ is given $\mathcal{L}_{\mathsf{Query}}(\Omega, q_i)$ and $\mathcal{A}$ and $\mathcal{S}$ execute a simulation of* $\mathsf{distQuery}$ *with $\mathcal{A}$ playing the role of the server and $\mathcal{S}$ playing the role of the client.*
- *$\mathcal{A}$ computes a bit b that is output by the experiment.*

$\mathbf{Real}_{\mathcal{A}}(1^k)$**:**

- *$\mathcal{A}$ outputs an oracle Ω, its approximation factor α and its error parameter ε.*
- *$\mathcal{C}$ computes $(K, \mathsf{EO}) \leftarrow \mathsf{Setup}(1^k, \Omega, \alpha, \varepsilon)$ and sends the encrypted graph* EO *to $\mathcal{A}$.*
- *$\mathcal{A}$ generates a polynomial number of adaptively chosen queries $(q_1, \ldots, q_m)$. For each query q_i, $\mathcal{A}$ and $\mathcal{C}$ execute $\mathsf{distQuery}_{\mathcal{C},\mathcal{A}}\big((K, q), \mathsf{EO}\big)$.*
- *$\mathcal{A}$ computes a bit b that is output by the experiment.*

We say that Graph *is adaptively $(\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$-semantically secure if for all ppt adversaries $\mathcal{A}$, there exists a ppt simulator $\mathcal{S}$ such that*

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

The definition above captures the fact that, given the encrypted oracle and its view of the query protocol, an adversarial server cannot learn any information about the oracle beyond the leakage.

4.2 Leakage

All the distance oracle encryption schemes we discuss in this work leak information. We describe and formalize these leakages below.

Setup leakage. The setup leakage of our first and second constructions, $\mathsf{GraphEnc}_1$ and $\mathsf{GraphEnc}_2$ in Sections 5.1 and 5.2, includes the total number of nodes in the underlying graph n, the maximum sketch size $S = \max_{v \in V} |\mathsf{Sk}_v|$ and the maximum distance over all seeds $D = \max_{v \in V} \max_{(w,\delta) \in \mathsf{Sk}_v} \delta$. The setup leakage of our third construction, $\mathsf{GraphEnc}_3$ in Section 5.3, includes n, S, D and the total number of seeds $Z = \sum_{v \in V} |\mathsf{Sk}_v|$.

Query pattern leakage. The query leakage of our first two constructions, $\mathsf{GraphEnc}_1$ and $\mathsf{GraphEnc}_2$, reveals whether the nodes in the query have appeared before. We refer to this as the *query pattern leakage* and formalize it below.

Definition 4.3 (Query pattern) *For two queries q, q' define $Sim(q, q') = (u = u', u = v', v = u', v = v')$, i.e., whether each of the nodes $q = (u, v)$ matches each of the nodes of $q' = (u', v')$. Let $\mathbf{q} = (q_1, \ldots, q_m)$ be a non-empty sequence of queries. Every query $q_i \in \mathbf{q}$ specifies a pair of nodes u_i, v_i. The query pattern leakage function $\mathcal{L}_{QP}(\mathbf{q})$ returns an $m \times m$ (symmetric) matrix with entry i, j equals $Sim(q_i, q_j)$. Note that $\mathcal{L}_{QP}$ does not leak the identities of the queried nodes.*

We do not claim that it is always reasonable for a graph encryption scheme to leak the query pattern - it may convey sensitive information in some settings. Furthermore, Definition 4.2 does not attempt to capture all possible leakages. As with many similar definitions, it does not capture side channels, and, furthermore, it does not capture leakage resulting from the client's behavior given the query answers, which, in turn may be affected by the choice of an approximation algorithm (see also [FIM+06, HKKN01] for a discussion of privacy of approximation algorithms).

Sketch pattern leakage. Our third construction, $\mathsf{GraphEnc}_3$, leaks the query pattern and an additional pattern we refer to as the *sketch pattern*. The sketch pattern reveals which seeds are shared between the different sketches of the oracle and the size of the sketches. We formalize this below by revealing randomized "pseudo-ids" of the seeds in each sketch.

Definition 4.4 (Sketch pattern leakage) *The sketch pattern leakage function $\mathcal{L}_{SP}(\Omega_G, q)$ for a graph G and a query $q = (u, v)$ is a pair (X, Y), where $X = \{f(w) : (w, \delta) \in \mathsf{Sk}_u\}$ and $Y = \{f(w) : (w, \delta) \in \mathsf{Sk}_v\}$ are* multi-sets *and $f : \{0,1\}^{\log n} \rightarrow \{0,1\}^{\log n}$ is a random function.*

It is not clear what this leakage implies in practice but we note that the leakage is not (directly) over the graph but over the *sketches* which contain a random subset of nodes. Therefore, it may be possible to add some form of noise in the sketches (e.g., using fake sketch elements) to guarantee some level of privacy to the original graph. We note that leakage is revealed in all SSE constructions such as [SWP00, CM05, Goh03, CGKO06, CK10, KPR12, KO12, KP13, CJJ+13, NPG14, CJJ+14]. However, in all these constructions the leakage is over a data structure (e.g., an inverted index) that holds *all* of the original data (i.e., all the keywords and documents). In our case, the leakage is over a structure that holds only a random subset of the data. This could provide additional help with respect to privacy but this is a topic for future work and is not the main focus of this paper.

4.3 Efficiency

We evaluate the efficiency and practicality of our constructions according to the following criteria:

- *Setup time:* the time for the client to pre-process and encrypt the graph;
- *Space complexity:* the size of the encrypted graph;
- *Query time:* The time to execute a shortest distance query on the encrypted graph;
- *Communication complexity:* the number of bits exchanged during a query operation.

5 Our Constructions

In this section, we describe our three oracle encryption schemes. The first scheme, $\mathsf{GraphEnc}_1$, is computationally efficient, but has high communication overhead. Our second scheme, $\mathsf{GraphEnc}_2$, is communication efficient but has high space overhead. Our third scheme, $\mathsf{GraphEnc}_3$, is computationally efficient with optimal communication complexity. $\mathsf{GraphEnc}_1$ and $\mathsf{GraphEnc}_2$ do not leak anything besides the Query Pattern, and $\mathsf{GraphEnc}_3$ also leaks the Sketch Pattern.

5.1 A Computationally-Efficient Scheme

We now describe our first scheme which is quite practical. The scheme $\mathsf{GraphEnc}_1 = (\mathsf{Setup}, \mathsf{distQuery})$ makes use of a symmetric-key encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and a PRP P. The Setup algorithm works as follows. Given a $1^k, \Omega_G$, α and ε:

- It pads each sketch to the maximum sketch size S by filling them with dummy values.
- It then generates keys K_1, K_2 for the encryption scheme and PRP respectively and sets $K = (K_1, K_2)$. For all $v \in V$, it computes a label $P_{K_2}(v)$ and creates an encrypted sketch $\mathsf{ESk}_v = (c_1, ..., c_\lambda)$, where $c_i \leftarrow \mathsf{Enc}_{K_1}(w_i \| \delta_i)$ is a symmetric-key encryption of the ith pair (w_i, δ_i) in Sk_v.
- It then sets up a dictionary DX in which it stores, for all $v \in V$, the pairs $(P_{K_2}(v), \mathsf{ESk}_v)$, ordered by the labels. The encrypted graph is then simply $\mathsf{EO} = \mathsf{DX}$.

The $\mathsf{distQuery}$ protocol works as follows. To query EO on $q = (u, v)$, the client sends a token $\mathsf{tk} = (\mathsf{tk}_1, \mathsf{tk}_2) = (P_{K_2}(u), P_{K_2}(v))$ to the server which returns the pair $\mathsf{ESk}_u := \mathsf{DX}[\mathsf{tk}_1]$ and $\mathsf{ESk}_v := \mathsf{DX}[\mathsf{tk}_2]$. The client then decrypts each encrypted sketch and computes $\min_{s \in \mathbf{I}} \mathsf{dist}(u, s) + \mathsf{dist}(s, v)$ (note that the algorithm only needs the sketches of the nodes in the query).

Security and efficiency. It is straightforward to see that the scheme is adaptively $(\mathcal{L}, \mathcal{L}_{QP})$-semantically secure, where $\mathcal{L}$ is the function that returns n, S and D. We defer a formal proof to the full version of this work. The communication complexity of the $\mathsf{distQuery}$ protocol is linear in S, where S is the maximum sketch size. Note that even though S is sub-linear in n, it could still be large in practice. For example, in the Das Sarma *et al.* construction $S = O(n^{2/\alpha} \cdot \log n)$. Also, in the case of multiple concurrent queries, this could be a significant bottleneck for the scheme.

In the following Section, we show how to achieve a solution with $O(1)$ communication complexity and in Section 6 we experimentally show that it scales to graphs with millions of nodes.

5.2 A Communication-Efficient Scheme

We now describe our second scheme $\mathsf{GraphEnc}_2 = (\mathsf{Setup}, \mathsf{distQuery})$ which is less computationally efficient than our first but is optimal with respect to communication complexity.

The details of the construction are given in Algorithms 1 and 2. It makes use of a SWHE scheme $\mathsf{SWHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, a pseudo-random permutation P and a family of universal hash functions $\mathcal{H}$.

The Setup algorithm works as follows. Given 1^k, Ω_G, α, and ε as inputs, it generates a public/secret-key pair $(\mathsf{pk}, \mathsf{sk})$ for SWHE. Let D be the maximum distance over all the sketches and S be the maximum sketch size. Setup sets $N := 2 \cdot D + 1$ and samples a hash function $h \xleftarrow{\$} \mathcal{H}$ with domain V and co-domain $[t]$, where $t = 2 \cdot S^2 \cdot \varepsilon^{-1}$.

It then creates a hash table for each node $v \in V$. More precisely, for each node v, it processes each pair $(w_i, \delta_i) \in \mathsf{Sk}_v$ and stores $\mathsf{Enc}_{\mathsf{pk}}(2^{N-\delta_i})$ at location $h(w_i)$ of a t-size array T_v. In other words, for all $v \in V$, it creates an array T_v such that for all $(w_i, \delta_i) \in \mathsf{Sk}_v$, $\mathsf{T}_v[h(w_i)] \leftarrow \mathsf{Enc}_{\mathsf{pk}}(2^{N-\delta_i})$. It then fills the empty cells of T_v with homomorphic encryptions of 0 and stores each hash table T_{v_1} through T_{v_n} in a dictionary DX by setting, for all $v \in V$, $\mathsf{DX}[P_K(v)] := \mathsf{T}_v$. Finally, it outputs DX as the encrypted oracle EO.

Fig. 2 below provides an example of one of the hash tables T_v generated from a sketch $\mathsf{Sk}_v = \{(w_1, \delta_1), \ldots, (w_s, \delta_s)\}$, where s is the size of the sketch. For all $i \in [s]$, the ciphertext

Algorithm 1: Setup algorithm for $\mathsf{GraphEnc}_2$

Input : $1^k, \Omega_G, \alpha, \varepsilon$
Output: EO

begin Setup
  Sample $K \xleftarrow{\$} \{0,1\}^k$;
  Initialize a dictionary DX;
  Generate a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SWHE.Gen}(1^k)$;
  Set $S := \max_{v \in V} |\mathsf{Sk}_v|$;
  Set $D := \max_{v \in V} \left\{ \max_{(w,\delta) \in \mathsf{Sk}_v} \delta \right\}$;
  Set $N := 2 \cdot D + 1$ and $t = 2 \cdot S^2 \cdot \varepsilon^{-1}$;
  Sample a hash function $h : V \rightarrow [t]$ from $\mathcal{H}$;
  foreach $v \in V$ **do**
    compute $\ell_v := P_K(v)$;
    initialize an array T_v of size t;
    foreach $(w_i, \delta_i) \in \mathsf{Sk}_v$ **do**
      set $\mathsf{T}_v[h(w_i)] \leftarrow \mathsf{SWHE.Enc}_{\mathsf{pk}}(2^{N-\delta_i})$;
      fill remaining cells of T_v with encryptions of 0; set $\mathsf{DX}[\ell_v] := \mathsf{T}_v$;
    end
  end
  Output K and $\mathsf{EO} = \mathsf{DX}$
end

$\mathsf{Enc}_{\mathsf{pk}}(2^{N-\delta_i})$ is stored at location $h(w_i)$ of the table T_v. For example, we place $\mathsf{Enc}_{\mathsf{pk}}(2^{2-\delta_j})$ to $\mathsf{T}_v[h(w_j)]$ since $h(w_j) = 1$. Finally, all remaining locations of T_v are filled with SWHE encryptions of 0. Notice that, since we are using probabilistic encryption, the encryptions of 0 are different, and are indistinguishable from the encryptions of the other values.

	0	$h(w_i)$	2		$h(w_j)$	$h(w_k)$	t-1
T_v	$Enc_{\mathrm{pk}}(0)$	$Enc_{\mathrm{pk}}(2^{N-\delta_i})$	$Enc_{\mathrm{pk}}(0)$	--	$Enc_{\mathrm{pk}}(2^{N-\delta_j})$	$Enc_{\mathrm{pk}}(2^{N-\delta_k})$	$Enc_{\mathrm{pk}}(0)$

Figure 2: One node's encrypted hash table.

The $\mathsf{distQuery}$ protocol works as follows. Given a query $q = (u, v)$, the client sends tokens $(\mathsf{tk}_1, \mathsf{tk}_2) = (P_K(u), P_K(v))$ to the server which uses them to retrieve the hash tables of nodes u and v by computing $\mathsf{T}_u := \mathsf{DX}[\mathsf{tk}_1]$ and $\mathsf{T}_v := \mathsf{DX}[\mathsf{tk}_2]$. The server then homomorphically evaluates an inner product over the hash tables. More precisely, it computes $c := \sum_{i=1}^{t} \mathsf{T}_u[i] \cdot \mathsf{T}_v[i]$, where $\sum$ and $\cdot$ refer to the homomorphic addition and multiplication operations of of the SWHE scheme. Finally, the server returns only c to the client who decrypts it and outputs $2N - \log_2(\mathsf{Dec}_{\mathsf{sk}}(c))$.

Note that the storage complexity at the server is $O(n \cdot t)$ and the communication complexity of $\mathsf{distQuery}$ is $O(1)$ since the server only returns a single ciphertext. In Section 5.2.1, we analyze the correctness and security of the scheme.

Remark. The reason we encrypt $2^{N-\delta_i}$ as opposed to δ_i is to make sure we can get the minimum sum over the distances from the sketches of both u and v. Our observation is that $2^x + 2^y$ is bounded by $2^{\max(x,y)+1}$. As we show Theorem 5.2, this approach does not, with high probability, affect the approximation factor from what the underlying distance oracle give us.

Algorithm 2: $\mathsf{DistQuery}$ algorithm for $\mathsf{GraphEnc}_2$

Input : Client's input is (K, q) and server's input is EO.
Output: Client's output is dist_q and server's output is $\perp$.

begin $\mathsf{distQuery}$
- C: client parses q as (u, v);
- $C \Rightarrow S$: client sends $\mathsf{tk} = (\mathsf{tk}_1, \mathsf{tk}_2) = (P_K(u), P_K(v))$;
- S: server retrieves $\mathsf{T}_1 := \mathsf{DX}[\mathsf{tk}_1]$ and $\mathsf{T}_2 := \mathsf{DX}[\mathsf{tk}_2]$;
- **foreach** $i \in [t]$ **do**
  - Server computes $c_i \leftarrow \mathsf{SWHE.Eval}(\times, \mathsf{T}_1[i], \mathsf{T}_2[i])$;
- **end**
- $S \Rightarrow C$: server sends $c \leftarrow \mathsf{SWHE.Eval}(+, c_1, \ldots, c_t)$;
- C: client computes $m \leftarrow \mathsf{SWHE.Dec}_{\mathsf{sk}}(c)$;
- C: client outputs $\mathsf{dist} = 2N - \log m$.

end

Instantiating & optimizing the SWHE scheme. For our experiments (see Section 6) we instantiate the SWHE scheme with the BGN construction of [BGN05]. We choose BGN due to the efficiency of its encryption algorithm and the compactness of its ciphertexts and keys (as compared to the lattice-based construction of [GHV10]). Unfortunately, the BGN *decryption* algorithm is expensive as it requires computations of discrete logarithms. To improve this, we make use of various optimizations. In particular, we compute discrete logs during decryption using the Baby step Giant step algorithm [Sha71] and use a pre-computed table to speed up the computation. We defer the details of our optimizations to the full version of this work.

5.2.1 Correctness

Here, we analyze the correctness of $\mathsf{GraphEnc}_2$. We first bound the collision probability of our construction and then proceed to prove correctness in Theorem 5.2 below.

Lemma 5.1 *Let $q = (u, v)$ be a shortest distance query and let $\mathcal{E}_q$ be the event that a collision occurred in the* Setup *algorithm while constructing the hash tables T_u and T_v. Then,* $\Pr[\,\mathcal{E}_q\,] \leq 2 \cdot \frac{S^2}{t}$.

Proof: Let Coll_v be the event that at least one collision occurs while creating v's hash table T_v (i.e., in Algorithm 1 Setup Line 13). Also, let $\mathsf{XColl}_{u,v}$ be the event that there exists at least one pair of distinct nodes $w_u \in \mathsf{Sk}_u$ and $w_v \in \mathsf{Sk}_v$ such that $h(w_u) = h(w_v)$. For any $q = (u, v)$, we have

$$\Pr[\,\mathcal{E}_q\,] \leq \Pr[\,\mathsf{Coll}_u\,] + \Pr[\,\mathsf{Coll}_v\,] + \Pr[\,\mathsf{XColl}_{u,v}\,]. \tag{1}$$

Let s_u be the size of Sk_u an s_v be the size of Sk_v. Since there are $\binom{s_u}{2}$ and $\binom{\mathsf{s}_v}{2}$ node pairs in Sk_u and Sk_v, respectively, and each pair collides under h with probability at most $1/t$, $\Pr[\,\mathsf{Coll}_u\,] \leq \frac{s_u^2}{2 \cdot t}$ and $\Pr[\,\mathsf{Coll}_v\,] \leq \frac{s_v^2}{2 \cdot t}$. On the other hand, if $\mathbf{I}$ is the set of common nodes in Sk_u and Sk_v, then $\Pr[\,\mathsf{XColl}_{u,v}\,] \leq \frac{(s_u - |\mathbf{I}|)(s_v - |\mathbf{I}|)}{t}$. Recall that $s_u = s_v \leq S$, so by combining with Eq. 1, we have $\Pr[\,\mathcal{E}_q\,] \leq 2 \cdot \frac{S^2}{t}$. ■

Note that in practice "intra-sketch" collision events Coll_u and Coll_v may or may not affect the correctness of the scheme. This is because the collisions could map the SWHE encryptions to locations that hold encryptions of 0 in other sketches. This means that at query time, these SWHE encryptions will not affect the inner product operation since they will be canceled out. Inter-sketch collision events $\mathsf{XColl}_{u,v}$, however, may affect the results since they will cause different nodes to appear in the intersection of the two sketches and lead to an incorrect sum.

Theorem 5.2 *Let $G = (V, E)$, $\alpha \geq 1$ and $\varepsilon < 1$. For all $q = (u, v) \in V^2$ with $u \neq v$,*

$$\Pr\left[\, \alpha \cdot \mathsf{dist}(u, v) - \log |\mathbf{I}| \leq d \leq \alpha \cdot \mathsf{dist}(u, v) \,\right] \geq 1 - \varepsilon,$$

where $(d, \perp) := \mathsf{GraphEnc_2.distQuery}\big((K, q), \mathsf{EO}\big)$, $(K, \mathsf{EO}) \leftarrow \mathsf{GraphEnc_2.Setup}(1^k, \Omega_G, \alpha, \varepsilon)$, and $\mathbf{I}$ is the number of common nodes between Sk_u and Sk_v.

Proof: Let $\mathbf{I}$ be the set of nodes in common between Sk_u and Sk_v and let $\mathsf{mindist} = \min_{w_i \in \mathbf{I}}\{\delta_i^u + \delta_i^v\}$, where for all $0 \leq i \leq |\mathbf{I}|$, $\delta_i^u \in \mathsf{Sk}_u$ and $\delta_i^v \in \mathsf{Sk}_v$. Note that at line 8 in Algorithm 2 $\mathsf{distQuery}$, the server returns to the client $c = \sum_{i=1}^{t} \mathsf{T}_u[i] \cdot \mathsf{T}_v[i]$.

Let $\mathcal{E}_q$ be the event a collision occurred during Setup in the construction of the hash tables T_u and T_v of u and v respectively. Conditioned on $\overline{\mathcal{E}_q}$, we therefore have that

$$\begin{aligned} c &= \sum_{i=1}^{|\mathbf{I}|} \mathsf{Enc_{pk}}(2^{N-\delta_i^u}) \cdot \mathsf{Enc_{pk}}(2^{N-\delta_i^v}) \\ &= \mathsf{Enc_{pk}}\big(2^{2N} \cdot \Sigma_{i=1}^{|\mathbf{I}|} 2^{-(\delta_i^u + \delta_i^v)}\big), \end{aligned}$$

where the first equality holds since for any node $w_i \notin \mathbf{I}$, one of the homomorphic encryptions $\mathsf{T}_u[i]$ or $\mathsf{T}_v[i]$ is an encryption of 0. It follows then that (conditioned on $\overline{\mathcal{E}_q}$) at Step 10 the client outputs

$$\begin{aligned} d &= 2N - \log\big(2^{2N} \cdot \Sigma_{i=1}^{|\mathbf{I}|} 2^{-(\delta_i^u + \delta_i^v)}\big) \\ &\leq 2N - \log\big(2^{2N - \mathsf{mindist}}\big) \\ &\leq \mathsf{mindist}, \end{aligned}$$

where the first inequality holds since $\mathsf{mindist} \leq (\delta_i^u + \delta_i^v)$ for all $i \in |\mathbf{I}|$. Towards showing a lower bound on d note that

$$\begin{aligned} d &= 2N - \log\big(2^{2N} \cdot \Sigma_{i=1}^{|\mathbf{I}|} 2^{-(\delta_i^u + \delta_i^v)}\big) \\ &\geq 2N - \log\big(2^{2N - \mathsf{mindist}} + |\mathbf{I}|\big) \\ &\geq \mathsf{mindist} - \log |\mathbf{I}|, \end{aligned}$$

where the first inequality also holds from $\mathsf{mindist} \leq (\delta_i^u + \delta_i^v)$ for all $i \in |\mathbf{I}|$. Now, by the (α, ε)-correctness of DO, we have that $\mathsf{mindist} \leq \alpha \cdot \mathsf{dist}(u, v)$ with probability at least $(1 - \varepsilon)$ over the coins of $\mathsf{DO.Setup}$. So, conditioned on $\overline{\mathcal{E}_q}$,

$$\mathsf{mindist} - \log |\mathbf{I}| \leq d \leq \alpha \cdot \mathsf{dist}(u, v).$$

The Theorem follows by combining this with Lemma 5.1 which bounds the probability of $\mathcal{E}_q$ and noting that Setup sets $t = 2 \cdot S^2 \cdot \varepsilon^{-1}$. ■

Space complexity. Note that to achieve (α, ε)-correctness, our construction produces encrypted sketches that are larger than the original sketches. More precisely, if the maximum sketch size of the underlying distance oracle is S, then the size of *every* encrypted sketch is $t = 2 \cdot S^2 \cdot \varepsilon^{-1}$, which is considerably larger. In Section 5.3, we describe a third construction which achieves better space efficiency at the cost of more leakage.

Remark on the approximation. Note that Theorem 5.2 also provides a lower bound of $\alpha \cdot \mathsf{dist}(u,v) - \log|\mathbf{I}|$ for the approximate distance. In particular, the bound depends on the set of common nodes $|\mathbf{I}|$ which varies for different queries but is small in practice. Furthermore, if $\log|\mathbf{I}|$ is larger than $\mathsf{mindist}$, the approximate distance returned could be negative (we indeed observe a few occurrences of this in our experiments).

To improve the accuracy of the approximation, one could increase the base in the homomorphic encryptions. More precisely, instead of using encryptions of the form $\mathsf{Enc}_{\mathsf{pk}}(2^{N-\delta})$ we could use $\mathsf{Enc}_{\mathsf{pk}}(B^{N-\delta})$ for $B = 3$ or $B = 4$. This would result in an improved lower bound of $\mathsf{mindist} - \log_B|\mathbf{I}|$ but would also increase the homomorphic decryption time since this increases the message space which in turn adds overhead to the decryption algorithm. We leave it as an open problem to further improve this lower bound without increasing the message space.

Remark on error rate. Given the above analysis, a client that makes γ queries will have an error ratio of $\varepsilon \cdot \gamma$. In our experiments we found that, in practice, when using the Das Sarma *et al.* oracle, setting $\sigma \approx 3$ results in a good approximation. So if we fix $\sigma = 3$ and set $t = O(\sqrt{n})$, then the error rate is $O\big(\gamma \cdot \log^2(n)/\sqrt{n}\big)$ which decreases significantly as n grows. In the case of the Cohen *et al.* all-distance sketch, if we fix $\rho = 4$ and set $t = O(\sqrt{n})$, then we achieve about the same error rate $O\big(\gamma \cdot \ln^2(n)/\sqrt{n}\big)$. We provide in Section 6 detailed experimental result on the error rate.

5.2.2 Security

In the following Theorem, we analyze the security of $\mathsf{GraphEnc}_2$.

Theorem 5.3 *If P is pseudo-random and SWHE is CPA-secure then $\mathsf{GraphEnc}_2$, as described above, is adaptively $(\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$-semantically secure, where $\mathcal{L}_{\mathsf{Setup}}(\Omega_G) = (n, S, D)$ and $\mathcal{L}_{\mathsf{Query}}(\Omega_G, q) = \mathcal{L}_{QP}(\Omega_G, q)$.*

Proof Sketch: Consider the simulator $\mathcal{S}$ that works as follows. Given leakage $\mathcal{L}_{\mathsf{Setup}}(\Omega_G) = (S, D)$, it starts by generating $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SWHE.Gen}(1^k)$. For all $1 \le i \le n$, it then samples $\ell_i \xleftarrow{\$} \{0,1\}^{\log n}$ without repetition and sets $\mathsf{DX}[\ell_i] := \mathtt{T}_i$, where $\mathtt{T}_i$ is an array that holds $t = 2 \cdot S_m^2 \cdot \varepsilon^{-1}$ homomorphic encryptions of $0 \in 2^N$, where $N = 2 \cdot D + 1$. It outputs $\mathsf{EO} = \mathsf{DX}$.

Given leakage $\mathcal{L}_{\mathsf{Query}}(\Omega_G, q) = \mathcal{L}_{QP}(\Omega_G, q)$ it checks if either of the query nodes u or v appeared in any previous query. If u appeared previously, $\mathcal{S}$ sets tk_1 to the value that was previously used. If not, it sets $\mathsf{tk}_1 := \ell_i$ for some previously unused ℓ_i. It does the same for the query node v; that is, it sets tk_2 to be the previously used value if v was previously queried or to an unused ℓ_i if it was not.

The theorem follows from the pseudo-randomness of P and the CPA-security of SWHE. ■

5.3 A Space-Efficient Construction

Although our second construction, $\mathsf{GraphEnc}_2$, achieves optimal communication complexity, it has two limitations. The first is that it is less computationally-efficient than our first construction $\mathsf{GraphEnc}_1$ both with respect to constructing the encrypted graph and to querying it. The second limitation is that its storage complexity is relatively high; that is, it produces encrypted graphs that are larger than the ones produced by $\mathsf{GraphEnc}_1$ by a factor of $2 \cdot S \cdot \varepsilon^{-1}$. These limitations are mainly due to the need to fill the hash tables with many homomorphic encryptions of 0. This also slows down the query algorithm since it has to homomorphically evaluate an inner product on two large tables.

To address this, we propose a third construction $\mathsf{GraphEnc}_3 = (\mathsf{Setup}, \mathsf{distQuery})$ which is both space-efficient and achieves $O(1)$ communication complexity. The only trade-off is that it leaks more than the two previous constructions.

Algorithm 3: Setup algorithm for $\mathsf{GraphEnc}_3$

Input : $1^k, \Omega_G, \alpha, \varepsilon$
Output: EO

begin Setup
  Sample $K_1, K_2 \xleftarrow{\$} \{0,1\}^k$;
  Initialize a counter $\mathsf{ctr} = 1$;
  Let $Z = \sum_{v \in V} |\mathsf{Sk}_v|$;
  Sample a random permutation π over $[Z]$;
  Initialize an array $\mathtt{Arr}$ of size Z;
  Initialize a dictionary DX of size n;
  Generate $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SWHE.Gen}(1^k)$;
  Set $S := \max_{v \in V} |\mathsf{Sk}_v|$;
  Set $D := \max_{v \in V} \left\{ \max_{(w,\delta) \in \mathsf{Sk}_v} \delta \right\}$;
  Set $N := 2 \cdot D + 1$ and $t = 2 \cdot S^2 \cdot \varepsilon^{-1}$;
  Initialize collision-resistant hash function $h : V \to [t]$;
  foreach $v \in V$ **do**
    sample $K_v \leftarrow \{0,1\}^k$;
    foreach $(w_i, \delta_i) \in \mathsf{Sk}_v$ **do**
      compute $c_i \leftarrow \mathsf{SWHE.Enc}_{\mathsf{pk}}(2^{N-\delta_i})$;
      if $i \neq |\mathsf{Sk}_v|$ **then**
        Set $\mathtt{N}_i = \langle h(w_i) \| c_i \| \pi(\mathsf{ctr}+1) \rangle$;
      else
        Set $\mathtt{N}_i = \langle h(w_i) \| c_i \| \mathsf{NULL} \rangle$;
      end
      Sample $r_i \xleftarrow{\$} \{0,1\}^k$;
      Set $\mathtt{Arr}[\pi(\mathsf{ctr})] := \langle \mathtt{N}_i \oplus H(K_v \| r_i), r_i \rangle$;
      Set $\mathsf{ctr} = \mathsf{ctr} + 1$;
    end
  end
  foreach $v \in V$ *(in random order)* **do**
    Set $\mathsf{DX}[P_{K_1}(v)] := \langle \mathsf{addr}_{\mathtt{Arr}}(\mathsf{h}_v) \| K_v \rangle \oplus F_{K_2}(v)$
  end
  Output $K = (K_1, K_2, \mathsf{pk}, \mathsf{sk})$ and $\mathsf{EO} = (\mathsf{DX}, \mathtt{Arr})$;
end

The details of the scheme are given in Algorithms 3 and 4. At a high-level, the scheme works similarly to $\mathsf{GraphEnc}_2$ with the exception that the encrypted sketches do not store encryptions of 0's, i.e., they only store the node/distance pairs of the sketches constructed by the underlying distance oracle. Implementing this high-level idea is not straightforward, however, because simply removing the encryptions of 0's from the encrypted sketches/hash tables reveals the size of the underlying sketches to the server which, in turn, leaks structural information about the graph. We overcome this technical difficulty by adapting a technique from [CGKO06] to our setting. Intuitively, we view the seed/distance pairs in each sketch Sk_v as a linked-list where each node stores a seed/distance pair. We then randomly shuffle all the nodes and place them in an array; that is, we place each node of each list at a random location in the array while updating the pointers so that the "logical" integrity of the lists are preserved (i.e., given a pointer to the head of a list we can still find all its nodes). We then encrypt all the nodes with a per-list secret key.

The scheme makes use of a SWHE scheme $\mathsf{SWHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$, a pseudo-random permutation P, a pseudo-random function F, a random oracle H and a collision-resistant hash function h modeled as a random function

The Setup algorithm takes as input a security parameter k, an oracle Ω_G , an approximation factor α, and an error parameter $\varepsilon < 1$. As shown in Algorithm 3, it first initializes a counter $\mathsf{ctr} = 1$ and samples a random permutation π over the domain $[Z]$, where $Z = \sum_{v \in V} |\mathsf{Sk}_v|$. It then initializes an Z-size array $\mathtt{Arr}$. It proceeds to create an encrypted sketch ESk_v from each sketch Sk_v as follows. It first samples a symmetric key K_v for this sketch. Then for each seed/distance pair (w_i, δ_i) in Sk_v, it creates a linked-list node $\mathtt{N}_i = \langle h(w_i) \| c_i \| \pi(\mathsf{ctr} + 1) \rangle$, where $c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(2^{N-\delta_i})$, and stores an H-based encryption $\langle N_i \oplus H(K_v \| r_v), r_v \rangle$ of the node at location $\pi(\mathsf{ctr})$ in $\mathtt{Arr}$. For the last seed/distance pair, it uses instead a linked-list node of the form $\mathtt{N}_i = \langle h(w_i) \| c_i \| \mathsf{NULL} \rangle$, it then increments ctr.

Setup then creates a dictionary DX where it stores for each node $v \in V$, the pair $(P_{K_1}(v), \langle \mathsf{addr}_{\mathtt{Arr}}(\mathsf{h}_v) \| K_v \rangle \oplus F_{K_2}(v))$, where $\mathsf{addr}_{\mathtt{Arr}}(\mathsf{h}_v)$ is the location in $\mathtt{Arr}$ of the head of v's linked-list. Figure 3 provides a detailed example for how we encrypt the sketch. Suppose node u's sketch Sk_u has the element $(a, d_1), (b, d_2), (c, d_3)$. The locations $\mathsf{ind1}, \mathsf{ind2}, \mathsf{ind3}$ in $\mathtt{Arr}$ are computed according the random permutation π.

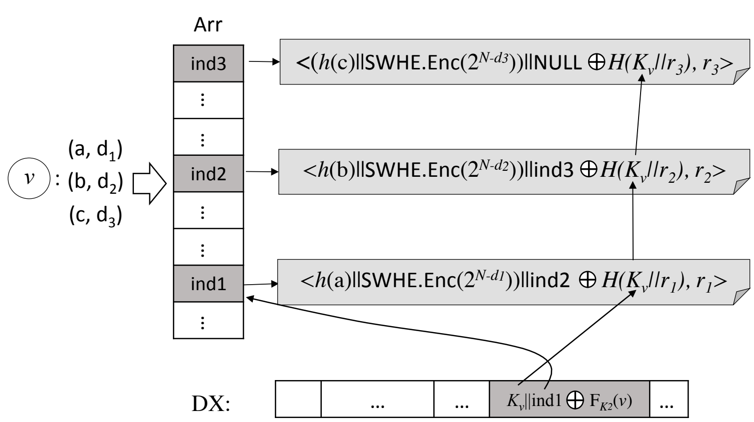


Figure 3: Example of encrypting $\mathsf{Sk}_u = \{(a, d_1), (b, d_2), (c, d_3)\}$.

The $\mathsf{distQuery}$ protocol, which is shown in Algorithm 4, works as follows. Given a query $q = (u, v)$, the client sends tokens $(\mathsf{tk}_1, \mathsf{tk}_2, \mathsf{tk}_3, \mathsf{tk}_4) = (P_{K_1}(u), P_{K_1}(v), F_{K_2}(u), F_{K_2}(v))$ to the server which uses them to retrieve the values $\gamma_1 := \mathsf{DX}[\mathsf{tk}_1]$ and $\gamma_2 := \mathsf{DX}[\mathsf{tk}_2]$. The server computes $\langle a_1 || K_u \rangle := \gamma_1 \oplus \mathsf{tk}_3$ and $\langle b_1 || K_v \rangle := \gamma_2 \oplus \mathsf{tk}_4$. Next, it recovers the lists pointed to by a_1 and b_1. More precisely, starting with $i = 1$, it parses $\mathtt{Arr}[a_1]$ as $\langle \sigma_u, r_u \rangle$ and decrypts σ_u by computing $\langle h_i \| c_i \| a_{i+1} \rangle := \sigma_u \oplus H(K_u \| r_u)$ while $a_{i+1} \neq \mathsf{NULL}$. And starting with $j = 1$, it does the same to recover $\langle h'_j \| c'_j \| b_{j+1} \rangle$ while $b_{j+1} \neq \mathsf{NULL}$.

The server then homomorphically computes an inner product over the ciphertexts with the same hashes. More precisely, it computes $\mathsf{ans} := \sum_{(i,j):h_i = h'_j} c_i \cdot c'_j$, where $\sum$ and $\cdot$ refer to the homomorphic addition and multiplication operations of the SWHE scheme. Finally, the server returns only ans to the client which decrypts it and outputs $2N - \log_2 (\mathsf{SWHE.Dec}_{\mathsf{sk}}(\mathsf{ans}))$.

Algorithm 4: The protocol $\mathsf{distQuery}_{C,S}$.

Input : Client's input is $K, q = (u, v)$ and server's input is EO
Output: Client's output is d and server's output is $\perp$

begin distQuery
  C: computes $(\mathsf{tk}_1, \mathsf{tk}_2, \mathsf{tk}_3, \mathsf{tk}_4) = (P_{K_1}(u), P_{K_1}(v), F_{K_2}(u), F_{K_2}(v))$;
  $C \Rightarrow S$: sends $\mathsf{tk} = (\mathsf{tk}_1, \mathsf{tk}_2, \mathsf{tk}_3, \mathsf{tk}_4)$;
  S: computes $\gamma_1 \leftarrow \mathsf{DX}[\mathsf{tk}_1]$ and $\gamma_2 \leftarrow \mathsf{DX}[\mathsf{tk}_2]$;
  if $\gamma_1 = \perp$ *or* $\gamma_2 = \perp$ **then**
    exit and return $\perp$ to the client
  end
  S: compute $\langle a_1 || K_u \rangle := \gamma_1 \oplus \mathsf{tk}_3$;
  S: parse $\mathtt{Arr}[a_1]$ as $\langle \sigma_u, r_u \rangle$;
  S: compute $\mathtt{N}_1 := \sigma_u \oplus H(K_u \| r_u)$;
  repeat
    parse $\mathtt{N}_i$ as $\langle h_i \| c_i \| a_{i+1} \rangle$;
    parse $\mathtt{Arr}[a_{i+1}]$ as $\langle \sigma_{i+1}, r_{i+1} \rangle$;
    compute $\mathtt{N}_{i+1} := \sigma_{i+1} \oplus H(K_u \| r_{i+1})$;
    set $i = i + 1$;
  until $a_{i+1} = \mathsf{NULL}$;
  S: compute $\langle b_1 || K_v \rangle := \gamma_2 \oplus \mathsf{tk}_4$;
  S: parse $\mathtt{Arr}[b_1]$ as $\langle \sigma_v, r_v \rangle$;
  S: compute $\mathtt{N}'_1 := \sigma_v \oplus H(K_v \| r_v)$;
  repeat
    parse $\mathtt{N}'_j$ as $\langle h'_j \| c'_j \| b_{j+1} \rangle$;
    parse $\mathtt{Arr}[b_{j+1}]$ as $\langle \sigma_{j+1}, r_{j+1} \rangle$;
    compute $\mathtt{N}'_{j+1} := \sigma_{j+1} \oplus H(K_v \| r_{j+1})$;
    set $j = j + 1$;
  until $b_{j+1} = \mathsf{NULL}$;
  S: set $s := \mathsf{SWHE.Enc}_{\mathsf{pk}}(0)$;
  foreach (N_i, N'_j) **do**
    if $h_i = h'_j$ **then**
      compute $p := \mathsf{SWHE.Eval}(\times, c_i, c'_j)$;
      compute $s := \mathsf{SWHE.Eval}(+, s, p)$;
    end
  end
  $S \Rightarrow C$: send s;
  C: compute $d := \mathsf{SWHE.Dec}_{\mathsf{sk}}(s)$
end

Note that the storage complexity at the server is $O(m + |V|)$ and the communication complexity of distQuery is still $O(1)$ since the server only returns a single ciphertext.

5.3.1 Correctness and Security

The correctness of $\mathsf{GraphEnc}_3$ follows directly from the correctness of $\mathsf{GraphEnc}_2$. To see why, observe that: (1) the homomorphic encryptions stored in the encrypted graph of $\mathsf{GraphEnc}_3$ are the same as those in the encrypted graph produced by $\mathsf{GraphEnc}_2$ with the exception of the encryptions of 0; and (2) the output d of the client results from executing the same homomorphic operations as in $\mathsf{GraphEnc}_2$, with the exception of the homomorphic sums with 0-encryptions.

We note that $\mathsf{GraphEnc}_3$ leaks only a little more than the previous constructions. With respect to setup leakage it reveals, in addition to (n, S, D), the total number of seeds Z. Intuitively, for a query $q = (u, v)$, the query leakage consists the query pattern leakage in addition to: (1) which seed/distance pairs in the sketches Sk_u and Sk_v are the same; and (2) the size of these sketches. This is formalized in Definition 4.4 as the sketch pattern leakage $\mathcal{L}_{SP}(\Omega_G, q)$. In the following Theorem, we summarize the security of $\mathsf{GraphEnc}_3$.

Theorem 5.4 *If P and F are pseudo-random, if* SWHE *is CPA-secure then* $\mathsf{GraphEnc}_3$*, as described above, is adaptively $(\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Query}})$-semantically secure in the random oracle model, where $\mathcal{L}_{\mathsf{Setup}}(\Omega_G) = (n, S, D, Z)$ and $\mathcal{L}_{\mathsf{Query}}(\Omega_G, q) = (\mathcal{L}_{QP}(\Omega_G, q), \mathcal{L}_{SP}(\Omega_G, q))$.*

Proof Sketch: Consider the simulator $\mathcal{S}$ that works as follows. Given leakage $\mathcal{L}_{\mathsf{Setup}} = (n, S, D, Z)$, for all $1 \leq i \leq Z$ it samples $\Gamma_i \xleftarrow{\$} \{0,1\}^{\log t + g(N) + \log Z + k}$, where $g(\cdot)$ is the ciphertext expansion of SWHE, $t = 2 \cdot S^2 \cdot \varepsilon^{-1}$ and $N = 2 \cdot D + 1$. It then stores all the Γ_i's in a Z-element array $\mathtt{Arr}$. For all $1 \leq i \leq n$, it samples $\ell_i \xleftarrow{\$} \{0,1\}^{\log n}$ without repetition and sets $\mathsf{DX}[\ell_i] \xleftarrow{\$} \{0,1\}^{\log Z + k}$. Finally, it outputs $\mathsf{EO} = (\mathsf{DX}, \mathtt{Arr})$.

Given leakage $\mathcal{L}_{\mathsf{Query}}(G, q) = (\mathcal{L}_{QP}(G, q), \mathcal{L}_{SP}(G, q))$ such that $\mathcal{L}_{SP}(G, q) = (X, Y)$, $\mathcal{S}$ first checks if either of the query nodes u or v appeared in any previous query. If u appeared previously, $\mathcal{S}$ sets tk_1 and tk_3 to the values that were previously used. If not, it sets $\mathsf{tk}_1 := \ell_i$ for some previously unused ℓ_i and tk_3 as follows. It chooses a previously unused $\alpha \in [Z]$ at random, a key $K_u \xleftarrow{\$} \{0,1\}^k$ and sets $\mathsf{tk}_3 := \mathsf{DX}[\mathsf{tk}_1] \oplus \langle \alpha \| K_u \rangle$. It then remembers the association between K_u and X and the sketch size $|\mathsf{Sk}_u|$. It does the same for the query node v, sets tk_2 and tk_4 analogously and associates $|\mathsf{Sk}_v|$ and Y with the key K_v it chooses.

It simulates the random oracle H as follows. Given (K, r) as input, it checks to see if: (1) K has been queried before (in the random oracle); and (2) if any entry in $\mathtt{Arr}$ has the form $\langle s, r \rangle$ where s is a $(\log t + g(N) + \log Z)$-bit string. If K has not been queried before, it initializes a counter $\mathsf{ctr}_K := 0$. If an appropriate entry exists in $\mathtt{Arr}$, it returns $s \oplus \langle \gamma, c, p \rangle$, where γ is the ctr^{th} element of the multi-set X or Y associated with K, c is a SWHE encryption of 0 and p is an unused address in $\mathtt{Arr}$ chosen at random or $\emptyset$ if $\mathsf{ctr} = |\mathsf{Sk}|$, where $|\mathsf{Sk}|$ is the sketch size associated with K. If no appropriate entry exists in $\mathtt{Arr}$, $\mathcal{S}$ returns a random value. The Theorem then follows from the pseudo-randomness of P and F and the CPA-security of SWHE. ■

6 Experimental Evaluation

In this section, we present experimental evaluations of our schemes on a number of large-scale graphs. We implement the Das Sarma *et al.* distance oracle ($\mathbf{DO}_1$) and Cohen *et al.* distance oracle ($\mathbf{DO}_2$) and all three of our graph encryption schemes. We use AES-128 in CBC mode for symmetric encryption and instantiate SWHE with the Boneh-Goh-Nissim (BGN) scheme, implemented in C++ with the Stanford Pairing-Based Library PBC[1]. We use OpenSSL[2] for all basic cryptographic primitives and use 128-bit security for all the encryptions. We use HMAC for PRFs and instantiate the hash function in $\mathsf{GraphEnc}_3$ with HMAC-SHA-256. All experiments were run on a 24-core 2.9GHz Intel Xeon, with 512 GBs of RAM running Linux.

[1] `http://crypto.stanford.edu/pbc/`

[2] `https://www.openssl.org/`

6.1 Datasets

We use real-world graph datasets publicly available from the Stanford SNAP website[3]. In particular, we use *as-skitter*, a large Internet topology graph; *com-Youtube*, a large social network based on the Youtube web site; *loc-Gowalla*, a location-based social network; *email-Enron*, an email communication network; and *ca-CondMat*, a collaboration network for scientific collaborations between authors of papers related to Condensed Matter research. Table 1 summarizes the main characteristics of these datasets.

Dataset	**Nodes**	**Edges**	**Diameter**	**Storage**
as-skitter	1,696,415	11,095,298	25	143MB
com-Youtube	1,134,890	2,987,624	20	37MB
loc-Gowalla	196,591	950,327	14	11MB
email-Enron	36,692	367,662	11	1.84MB
ca-CondMat	23,133	186,936	14	158KB

Table 1: The graph datasets used in our experiments

Notice that some of these datasets contain millions of nodes and edges and that the diameters of these graphs are small. This is something that has been observed in many real-life graphs [LKF05] and is true for expander and small-world graphs, which are known to model many real-life graphs. The implication of this, is that the maximum distance D in the sketches generated by the distance oracles is, in practice, small and therefore the value N that we use in $\mathsf{GraphEnc}_2$ and $\mathsf{GraphEnc}_3$ (see Algorithm 1 and 3) is typically small.

6.2 Overview

For a graph $G = (V, E)$ with n nodes, we summarize in Table 2 our constructions' space, setup, and communication complexities as well as the complexities for both the server and client during the query phase. Note that the complexities for each scheme also depend on α, however, in practice, since setting σ for $\mathbf{DO}_1$ (ρ for $\mathbf{DO}_2$) to some small numbers resulted good approximations, therefore, it makes $\alpha = O(\log n)$. In our experiments, we test different σ and ρ's and the sketch size, $|\mathsf{Sk}_v|$, for each node is sublinear in the size of the graph, i.e. $O(\log n)$.

Scheme	$\mathsf{GraphEnc}_1$	$\mathsf{GraphEnc}_2$	$\mathsf{GraphEnc}_3$
Space	$O(n \log n)$	$O(n \log^2 n/\varepsilon)$	$O(n \log n)$
Setup Time	$O(n \log n)$	$O(n \log^2 n/\varepsilon)$	$O(n \log n)$
Communication	$O(\log n)$	$O(1)$	$O(1)$
Server Query Comp.	$O(1)$	$O(\log^2 n/\varepsilon)$	$O(\log n)$
Client Query Comp.	$O(\log n)$	$O(diameter)$	$O(diameter)$

Table 2: The space, setup, communication, and query complexities of our constructions (α is set to be in $O(\log n)$).

Table 3 summarizes our experimental results. Compared to existing schemes, such as [ACM+13], our experiments shows that the constructions are very efficient and scalable for large real dataset. For example, in [ACM+13], it takes several minutes to securely compute the shortest path distance for graph with only tens to hundreds of nodes, whereas it takes only seconds for our scheme to query the encrypted graph up to 1.6 million nodes.

[3]`https://snap.stanford.edu/data/`

Dataset	sketch size S	Graph Sketching Scheme	$\mathsf{GraphEnc}_1$			$\mathsf{GraphEnc}_2$				$\mathsf{GraphEnc}_3$		
			Comm. per query (in bytes)	Setup Time per node (in ms)	Size per node (in KBs)	T size	Comm. per query (in bytes)	Setup Time per node (in secs)	Size per node (in MBs)	Comm. per query (in bytes)	Setup Time per node (in ms)	Size per node (in KBs)
As-skitter	80	$\mathbf{DO}_1$	3,840	16.7	1.94	11K	34	7.3	1.1	34	20.1	1.91
	71	$\mathbf{DO}_2$	3,120	14	1.63	8.4K	34	6.59	0.76	34	16	1.83
Youtube	80	$\mathbf{DO}_1$	3,840	16.5	1.94	10K	34	8	1.1	34	18.2	1.91
	68	$\mathbf{DO}_2$	3,120	14.5	1.63	8.5K	34	6.57	0.76	34	17.3	1.7
Gowalla	70	$\mathbf{DO}_1$	3360	14.9	1.7	7.5K	34	7.4	0.82	34	15.6	1.71
	53	$\mathbf{DO}_2$	2544	12	1.29	7K	34	5	0.62	34	14.7	1.41
Enron	60	$\mathbf{DO}_1$	2880	12.5	1.44	7K	34	5.6	0.76	34	14	1.48
	45	$\mathbf{DO}_2$	2160	9.39	1.11	6.5K	34	4.81	0.53	34	10	1.25
CondMat	55	$\mathbf{DO}_1$	2640	11.8	1.34	5.5K	34	4.65	0.65	34	13.2	1.31
	42	$\mathbf{DO}_2$	2016	7.8	1.03	5K	34	3.8	0.49	34	8.2	1.21

Table 3: A full performance summary for $\mathsf{GraphEnc}_1$, $\mathsf{GraphEnc}_2$, and $\mathsf{GraphEnc}_3$

6.3 Performance of $\mathsf{GraphEnc}_1$

We evaluate the performance of $\mathsf{GraphEnc}_1$ using both the Das Sarma *et al.* and Cohen *et al.* distance oracles. For the Das Sarma *et al.* oracle ($\mathbf{DO}_1$), we set the sampling parameter $\sigma = 3$ and for the Cohen *et al.* oracle ($\mathbf{DO}_2$) we set the rank parameter $\rho = 4$. We choose these parameters because they resulted in good approximation ratios and the maximum sketch sizes (i.e., S) of roughly the same amount. Note that, the approximation factor α in those then is in $O(\log n)$ for $\mathsf{GraphEnc}_1$, therefore, the communication complexity (see Table 2) in $\mathsf{GraphEnc}_1$ is $O(\log n)$. We can see from Table 3 that the time to setup an encrypted graph with $\mathsf{GraphEnc}_1$ is practical—even for large graphs. For example, it takes only 8 hours to setup an encryption of the *as-skitter* graph which includes 1.6 million nodes. Since the $\mathsf{GraphEnc}_1.\mathsf{Setup}$ is highly-parallelizable, we could speed setup time considerably by using a cluster. A cluster of 10 machines would be enough to bring the setup time down to less than an hour. Furthermore, the size of the encrypted sketches range from 1KB for *CondMat* to 1.94KB for *as-skitter* per node. The main limitation of this construction is that the communication is proportional to the size of the sketches. We tested for various sketch sizes, and the communication per query went up to 3.8KB for *as-skitter* when we set $S = 80$. This can become quite significant if the server is interacting with multiple clients.

6.4 Performance of $\mathsf{GraphEnc}_2$

The first column in Table 3 of the $\mathsf{GraphEnc}_2$ experiments gives the size the encrypted hash tables T_v constructed during $\mathsf{GraphEnc}_2.\mathsf{Setup}$. Table sizes range from 5K for *ca-CondMat* to $11K$ for *as-skitter*.

The Time column gives the time to create an encrypted hash-table/sketch per node. This includes generating the BGN encryptions of the distances and the 0-encryptions. Note that this makes $\mathsf{GraphEnc}_2.\mathsf{Setup}$ quite costly, about 3 orders of magnitude more expensive than $\mathsf{GraphEnc}_1.\mathsf{Setup}$. This is mostly due to generating the 0-encryptions. Note, however, that similarly to $\mathsf{GraphEnc}_1$, we can use extensive parallelization to speed up the setup. For example, using a cluster of 100 machines, we can setup the encrypted graph on the order of hours, even for *as-skitter* which includes 1.6 million nodes. The space overhead per node is also large, but the encrypted graph itself can be distributed in a cluster since every encrypted sketch is independent of the other. Finally, as shown in Table 3, $\mathsf{GraphEnc}_2$ achieves a constant communication cost of 34B.

In Fig. 4, we report on the intra- and inter-collisions that we observed when executing over 10K different queries over our datasets. The collision probability ranges between 1% and 3.5%. As we can see from the results, the oracle $\mathbf{DO}_2$ has less collisions than $\mathbf{DO}_1$.

We would like to point out that those inter-collisions can be *detected* by associating with each encryption of a node a random value and its inverse value that are unique for each node. If two different nodes collide, the product of these values will be a random value, whereas if the same node is mapped to the same entry the product will give 1. More discussion about this technique will appear in the full version of this work.

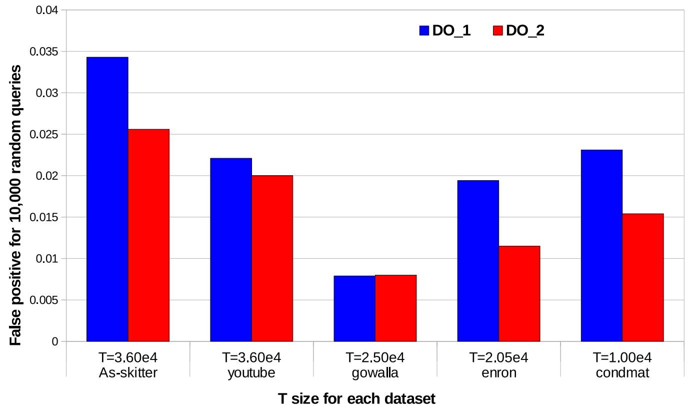


Figure 4: Collision probabilities for different datasets

6.5 Performance of $\mathsf{GraphEnc}_3$

The $\mathsf{GraphEnc}_3$ columns in Table 3 show that $\mathsf{GraphEnc}_3$ is as efficient as $\mathsf{GraphEnc}_1$ in terms of setup time and encrypted sketch size. Moreover, it achieves $O(1)$ communication of 34B like $\mathsf{GraphEnc}_2$. Using a single machine, $\mathsf{GraphEnc}_3.\mathsf{Setup}$ took less than 10 hours to encrypt *as-skitter*, but like the other schemes, it is highly parallelizable, and this could be brought down to an hour using 10 machines. We instantiated the hash function h using a cryptographic keyed hash function HMAC-SHA-256.

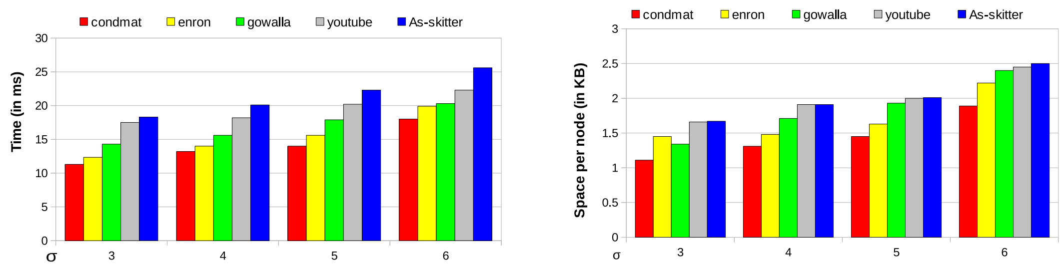
6.5.1 Construction time & encrypted sketch size

Since the performance of $\mathsf{GraphEnc}_3$ depends only on the size of the underlying sketches we investigate the relationship between the performance of $\mathsf{GraphEnc}_3.\mathsf{Setup}$ and the sampling and rank parameters of the Das Sarma *et al.* and Cohen *et al.* oracles, respectively. We use values of σ and ρ ranging from 3 to 6 in each case which resulted in maximum sketch sizes S ranging from 43 to 80. Figure 5 and Figure 6 give the construction time and size overhead of an encrypted sketch when using the Das Sarma *et al.* oracle and Cohen *et al.* oracle respectively.

In each case, the construction time scales linearly when σ and ρ increase. Also, unlike the previous schemes, $\mathsf{GraphEnc}_3$ produces encrypted sketches that are compact since it does not use 0-encryptions for padding purposes.

6.5.2 Query Time

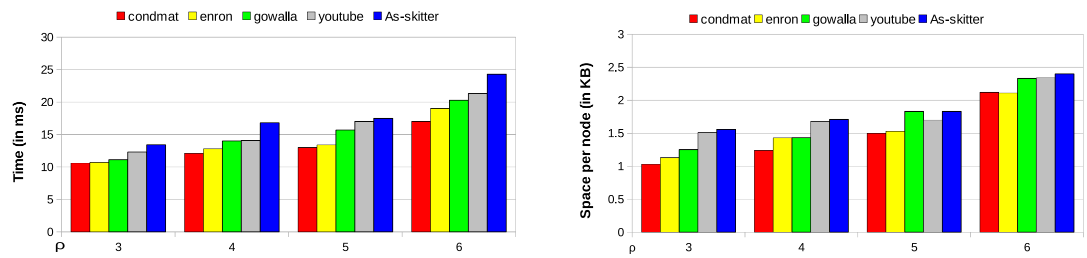
We measured the time to query an encrypted graph as a function of the oracle sampling/rank parameter. The average time at the server (taken over 10K random queries) is given in Figure 7 for all our graphs and using both distance oracles. In general, the results show that query time is fast and practical. For *as-skitter*, the query time ranges from 6.1 to 10 milliseconds with the Das Sarma *et al.* oracle and from 5.6 to 10 milliseconds with the Cohen *et al.* oracle. Query time is dominated by the homomorphic multiplication operation of the BGN scheme. But the number of multiplications only depends on the number of common seeds from the two encrypted sketches



Construction Time (in ms)

Size Overhead (in KB)

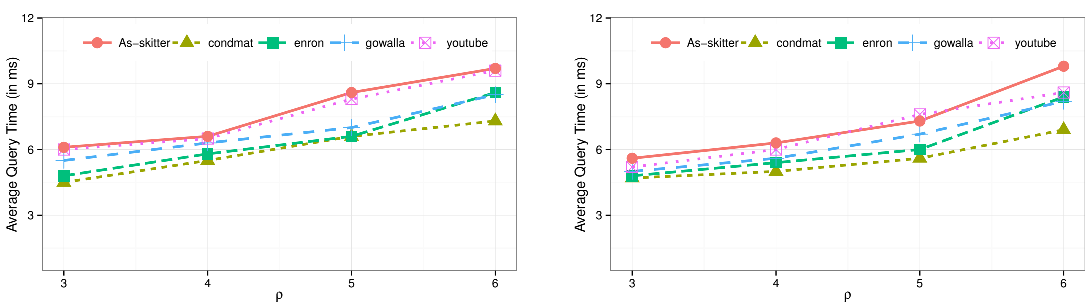
Figure 5: Construction time and size overhead ($\mathbf{DO}_1$)



Construction Time (in ms)

Size Overhead (in KB)

Figure 6: Construction time and size overhead ($\mathbf{DO}_2$)



(a) Query Time (in ms) using $\mathbf{DO}_1$

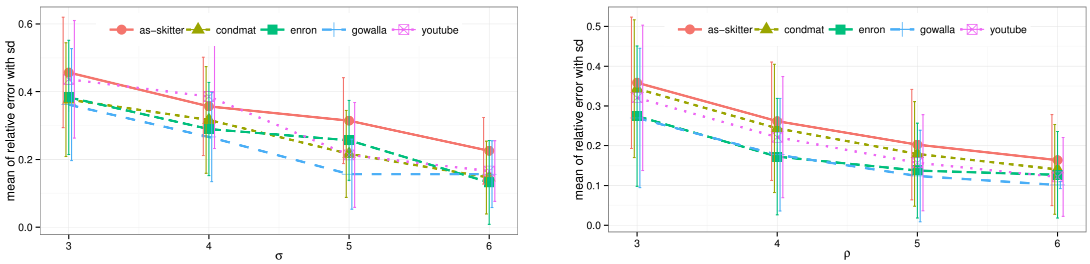
(b) Query Time (in ms) $\mathbf{DO}_2$

Figure 7: Average Query time

and, furthermore, these operations are independent so they can be parallelized. We note that we use mostly un-optimized implementations of all the underlying primitives and we believe that a more careful implementation (e.g., faster pairing library) would reduce the query time even further. We also measure the decryption time at the client. As pointed out previously, decryption time depends on N which itself is a function of the diameter of the graph. Since all our graphs have small diameter, client decryption time—which itself consists of a BGN decryption— was performed very efficiently. In particular, the average decryption time was less than 4 seconds and in most cases the decryption ranged between 1 and 3 seconds.

Finally, we would like to mention that there is some additional information that is leaked. In our construction, we leak the parameter ρ and σ that are related to the size of the encrypted graph and this may leak some information about how "hard" it is to approximate the shortest distance values for the particular graph at hand. Also, the time that it takes to estimate the final result at the client may reveal the diameter of the graph, since it is related to the N and the max distance in the sketches.

6.6 Approximation errors



DO$_1$ **DO**$_2$

Figure 8: Mean of Estimated Error with Standard Deviation

We investigate the approximation errors produced by our schemes. We generate 10K random queries and run the $\mathsf{Query}_{C,S}$ protocol. For client decryption, we recover $2N - \log m$ and round it to its floor value. We used breadth-first search (BFS) to compute the exact distances between each pair of nodes and we compare the approximate distance returned by our construction to exact distances obtained with BFS. We report the mean and the standard deviation of the relative error for each dataset. We used both oracles to compute the sketches. We present our results in Figure 8, which shows that our approximations are quite good. Indeed, our experiments show that our constructions could report *better* approximations than the underlying oracles. This is due to the fact that both oracles overestimate the distance so subtracting $\log |\mathbf{I}|$ can improve the approximation. For the *Gowalla* dataset, the mean of the relative error ranges from 0.36 to 0.13 when using the Das Sarma *et al.* oracle $\mathbf{DO}_1$. For *as-skitter*, it ranges from 0.45 to 0.22. The mean error and the variance decreases as we increase the size of each sketch. In addition, we note that $\mathbf{DO}_2$ performs better in all datasets. Also, half of the distances returned are exact and most of the distances returned are at most 2 away from the real distance. Figure 9 shows the histogram for the absolute error when using $\mathbf{DO}_2$ with $\rho = 3$. All the other datasets are very similar to them, so we omit them due to space limitations.

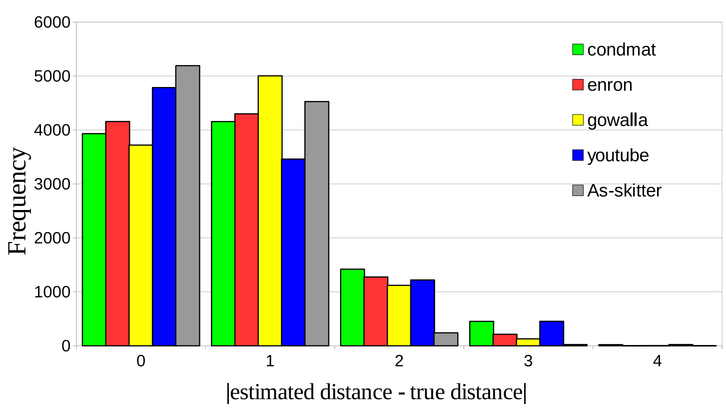


Figure 9: Absolute error histogram $\mathbf{DO}_2$ and $\rho = 3$

We note that a very small number of distances were negative and we removed them from the experiments. Negative distances result from the intersection size $|\mathbf{I}|$ being very large. Indeed, when the client decrypts the SWHE ciphertext returned by the server, it recovers $d \geq \mathsf{mindist} - \log |\mathbf{I}|$. If $|\mathbf{I}|$ is large and $\mathsf{mindist}$ is small (say, 1 or 2) then it is very likely that d is negative. However, in the experiments, the number of removed negative values were very small (i.e., 80 out of 10000 queries).

7 Conclusion

In this work, we described three graph encryption schemes that support approximate shortest distance queries. Our first solution, $\mathsf{GraphEnc}_1$, is based only on symmetric-key primitives and is computationally very efficient while our second solution, $\mathsf{GraphEnc}_2$, is based on somewhat homomorphic encryption and is optimal in terms of communication complexity. Furthermore, our third solution, $\mathsf{GraphEnc}_3$, achieves the "best of both worlds" and is computationally very efficient with optimal communication complexity. Our schemes work with any sketched-based distance oracle. We implemented our constructions and evaluated their efficiency experimentally, showing that our constructions are practical for large-scale graphs.

References

[ACM+13] Abdelrahaman Aly, Edouard Cuvelier, Sophie Mawet, Olivier Pereira, and Mathieu Van Vyve. Securely solving simple combinatorial graph problems. In *Financial Cryptography*, pages 239–257, 2013.

[BGN05] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC 2005*, pages 325–342, 2005.

[BSA13] Marina Blanton, Aaron Steele, and Mehrdad Aliasgari. Data-oblivious graph algorithms for secure computation and outsourcing. In *ASIACCS*, pages 207–218, 2013.

[CDF+13] Edith Cohen, Daniel Delling, Fabian Fuchs, Andrew V. Goldberg, Moisés Goldszmidt, and Renato F. Werneck. Scalable similarity estimation in social networks: closeness, node labels, and random edge lengths. In *COSN*, pages 131–142, 2013.

[CFL10] James Cheng, Ada Wai-Chee Fu, and Jia Liu. K-isomorphism: privacy preserving network publication against structural attacks. In *SIGMOD Conference*, pages 459–470, 2010.

[CGKO06] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *CCS 2006*, pages 79–88. ACM, 2006.

[Che14] Shiri Chechik. Approximate distance oracles with constant query time. In *STOC*, pages 654–663, 2014.

[CHKZ03] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003.

[CJJ+13] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO '13*, pages 353–373, 2013.

[CJJ+14] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[CK10] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT '10*, volume 6477, pages 577–594, 2010.

[CM05] Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS '05*, pages 442–455. Springer, 2005.

[Coh14] Edith Cohen. All-distances sketches, revisited: Hip estimators for massive graphs analysis. In *PODS*, pages 88–99, 2014.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[DSGNP10] A. Das Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *WSDM*, pages 401–410, 2010.

[FIM+06] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. *ACM Transactions on Algorithms*, 2(3):435–472, 2006.

[Gen09] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09*, pages 169–178. ACM Press, 2009.

[GHV10] C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple bgn-type cryptosystem from lwe. In *Advances in Cryptology - EUROCRYPT '10*, pages 506–522. Springer, 2010.

[GO96] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[Goh03] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216/`.

[GYJ+11] Jun Gao, Jeffrey Xu Yu, Ruoming Jin, Jiashuai Zhou, Tengjiao Wang, and Dongqing Yang. Neighborhood-privacy protected shortest distance computing in cloud. In *SIGMOD*, pages 409–420, 2011.

[HKKN01] Shai Halevi, Robert Krauthgamer, Eyal Kushilevitz, and Kobbi Nissim. Private approximation of np-hard functions. In *STOC*, pages 550–559. ACM, 2001.

[HLP+13] Wook-Shin Han, Sangyeon Lee, Kyungyeol Park, Jeong-Hoon Lee, Min-Soo Kim, Jinha Kim, and Hwanjo Yu. Turbograph: a fast parallel graph engine handling billion-scale graphs in a single PC. In *KDD*, pages 77–85, 2013.

[KG14] Aapo Kyrola and Carlos Guestrin. Graphchi-db: Simple design for a scalable graph database system - on just a PC. *CoRR*, abs/1403.0701, 2014.

[KL08] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* Chapman & Hall/CRC, 2008.

[KNRS13] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Analyzing graphs with node differential privacy. In *TCC*, pages 457–476, 2013.

[KO12] K. Kurosawa and Y. Ohtaki. Uc-secure searchable symmetric encryption. In *Financial Cryptography and Data Security (FC '12)*, Lecture Notes in Computer Science, pages 285–298. Springer, 2012.

[KP13] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security (FC '13)*, 2013.

[KPR12] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security (CCS '12)*. ACM Press, 2012.

[LGK+10] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new framework for parallel machine learning. In *UAI*, pages 340–349, 2010.

[LKF05] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.

[LT08] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *SIGMOD Conference*, pages 93–106, 2008.

[MAB+10] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010.

[MY12] Kyriakos Mouratidis and Man Lung Yiu. Shortest path computation with no information leakage. *PVLDB*, pages 692–703, 2012.

[NPG14] M. Naveed, M. Prabhakaran, and C. Gunter. Dynamic searchable encryption via blind storage. In *Oakland S& P 2014*, 2014.

[PBCG09] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. Fast shortest path distance estimation in large networks. In *CIKM*, pages 867–876, 2009.

[PWJ04] Natasa Przulj, Dennis A. Wigle, and Igor Jurisica. Functional topology in a network of protein interactions. *Bioinformatics*, 20(3):340–348, 2004.

[QXSW13] Zichao Qi, Yanghua Xiao, Bin Shao, and Haixun Wang. Toward a distance oracle for billion-node graphs. In *VLDB*, pages 61–72, 2013.

[RAD78] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–180, 1978.

[SEHK12] Mohamed Sarwat, Sameh Elnikety, Yuxiong He, and Gabriel Kliot. Horton: Online query execution engine for large distributed graphs. In *ICDE*, pages 1289–1292, 2012.

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. In *1969 Number Theory Institute*, pages 415–440. Providence, R.I., 1971.

[SPS14] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.

[SWL13] Bin Shao, Haixun Wang, and Yatao Li. Trinity: a distributed graph engine on a memory cloud. In *SIGMOD*, pages 505–516, 2013.

[SWP00] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *Oakland S & P*, pages 44–55, 2000.

[SY13] Entong Shen and Ting Yu. Mining frequent graph patterns with differential privacy. In *KDD 2013*, pages 545–553, 2013.

[TZ05] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, January 2005.

[WNL+14] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T.-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 215–226, 2014.