

# GRECS: Graph Encryption for Approximate Shortest Distance Queries

Xianrui Meng<sup>\*1</sup>, Seny Kamara<sup>†2</sup>, Kobbi Nissim<sup>‡3</sup>, and George Kollios<sup>§1</sup>

<sup>1</sup>Department of Computer Science, Boston University

<sup>2</sup>Microsoft Research

<sup>3</sup>Department of Computer Science, Ben-Gurion University

## 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 **En**Cryption 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 [31], GraphLab [30], Horton [38], Trinity [40], TurboGraph [22], and GraphChi-DB [27]) and industry (e.g., Neo4j, Titan, DEX, and GraphBase.) Furthermore, with the

---

\*xmeng@cs.bu.edu

†senyk@microsoft.com

‡kobbi@cs.bgu.ac.il

§gkollios@cs.bu.edu

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 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 [6] 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 [6] 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. Graph encryption is a special case of *structured encryption*, which is a encryption scheme that encrypts 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 [42, 5, 13, 45, 24, 23, 4, 3, 33, 43] 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 consider 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. Such shortest distance query is 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 [35] 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.

On the other hand, we note that 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* [44, 14, 10]; that is, given two vertices  $v_1$  and  $v_2$ , the structure returns a distance  $d$  that is at most  $\alpha \cdot \text{dist}(v_1, v_2) + \beta$ , where  $\alpha, \beta > 1$  and  $\text{dist}(v_1, v_2)$  is the exact distance between  $v_1$  and  $v_2$ .

Below we summarize our contributions for this paper:

- We propose three constructions. 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 graphs 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 [17, 29, 8, 12], use differential privacy [15] to query graph statistics privately [25, 41], or use private information retrieval (PIR) [32] to privately recover shortest paths. We note that none of these approaches are appropriate in our context where the graph itself stores sensitive information therefore it 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 [6]. Structured encryption is a generalization of searchable symmetric encryption (SSE) which was first proposed by [42]. The notion of adaptive semantic security was introduced by Curtmola, Garay, Kamara and Ostrovsky in [13] and generalized to the setting of structured encryption in [6].

Another approach to execute graph algorithms over encrypted and outsourced graphs is to use Oblivious RAM [20] 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 [46] presents the oblivious data structure for computing shortest path on planar graphs using ORAM. For a sparse planar graph with  $O(n)$  edges, their approach requires  $O(n^{1.5})$  space complexity, but at the cost of  $O(\sqrt{n} \log n)$  online query time. Other techniques such as those developed by Blanton, Steele and Aliasgari [1] are not practical for sparse graphs and do not scale to large graphs due to the complexity of the underlying secure operations which are instantiated with secure multi-party computation protocols.

**Distance oracles** Computing the 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 in logarithmic query time. Solutions such as [14, 34, 36, 10, 7, 9, 11] 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 [14] requires  $\tilde{O}(n^{1/c})$  work to return a  $(2c-1)$ -approximation of the shortest distance.

## 2 Preliminaries and Notations

**Notation** 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  $\text{dist}(u, v)$ . The notation  $[n]$  represents the set of integers  $\{1, \dots, n\}$ . We write  $x \leftarrow \chi$  to represent an element  $x$  being sampled from a distribution  $\chi$ . We write  $x \stackrel{\$}{\leftarrow} X$  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  $i^{\text{th}}$  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  $\nu : \mathbb{N} \rightarrow \mathbb{N}$  is negligible in  $k$  if for every positive polynomial  $p(\cdot)$  and all sufficiently large  $k, \nu(k) < 1/p(k)$ . We write  $f(k) = \text{poly}(k)$  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  $f(k) = \text{negl}(k)$  to mean that there exists a negligible function  $\nu(\cdot)$  such that  $f(k) \leq \nu(k)$  for all sufficiently large  $k$ .

**Data structures** A dictionary  $\text{DX}$  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. The insert operation takes as input a dictionary  $\text{DX}$  and a label/value pair  $(\ell, v)$  and adds the pair to  $\text{DX}$ . We denote this as  $\text{DX}[\ell] := v$ . A lookup operation takes as input a dictionary  $\text{DX}$  and a label  $\ell_i$  and returns the associated value  $v_i$ . We denote this as  $v_i := \text{DX}[\ell]$ . Dictionaries can be instantiated using hash tables and various kinds of search trees.

## 2.1 Cryptographic Tools

**Encryption and homomorphic 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  $\text{SKE} = (\text{Gen}, \text{Enc}, \text{Dec})$  is a set of three polynomial-time algorithms that work as follows.  $\text{Gen}$  is a probabilistic algorithm that takes a security parameter  $k$  as input and returns a secret key  $K$ ;  $\text{Enc}$  is a probabilistic algorithm that takes as input a key  $K$  and a message  $m$  and returns a ciphertext  $c$ ;  $\text{Dec}$  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  $\text{PKE} = (\text{Gen}, \text{Enc}, \text{Dec})$  is similarly defined except that  $\text{Gen}$  outputs a public/private key pair  $(\text{pk}, \text{sk})$  and  $\text{Enc}$  encrypts messages with the public key  $\text{pk}$ . 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 [26] for formal definitions of symmetric-key encryption scheme and CPA-security.

A public-key encryption scheme is homomorphic if, in addition to  $(\text{Gen}, \text{Enc}, \text{Dec})$  it also includes an evaluation algorithm  $\text{Eval}$  that takes as input a function  $f$  and a set of ciphertexts  $c_1 \leftarrow \text{Enc}_{\text{pk}}(m_1)$  through  $c_n \leftarrow \text{Enc}_{\text{pk}}(m_n)$  and returns a ciphertext  $c$  such that  $\text{Dec}_{\text{sk}}(c) = f(m_1, \dots, m_n)$ . If a homomorphic encryption scheme supports the evaluation of any polynomial-time function, then it is fully-homomorphic (FHE) [37, 18] otherwise it is somewhat homomorphic (SWHE). In this work, we make use of only “low degree” homomorphic encryption; namely, we only require the evaluation of quadratic polynomials. More precisely, we need a scheme that supports any number of additions and a single multiplication. After that, any number of additions are allowed but not multiplications. In particular, we need the evaluation algorithm to be additively homomorphic as follows,  $\text{Enc}_{\text{pk}}(m_1 + m_2) = \text{Eval}(+, \text{Enc}_{\text{pk}}(m_1), \text{Enc}_{\text{pk}}(m_2))$ . Moreover, we also need the evaluation algorithm to be multiplicative homomorphic, i.e.  $\text{Enc}_{\text{pk}}(m_1 m_2) = \text{Eval}(\times, \text{Enc}_{\text{pk}}(m_1), \text{Enc}_{\text{pk}}(m_2))$ . However, we cannot evaluate the multiplication of two message, if we have already evaluated it once, therefore, we can only homomorphically evaluate quadratic polynomials. Concrete instantiations of such schemes include the scheme of Boneh, Goh and Nissim (BGN) [2] based on bilinear maps and the scheme of Gentry, Halevi and Vaikuntanathan [19] 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 [26] for formal definitions of PRFs and PRPs.

### 3 Graph Encryption

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

**Definition 3.1 (Graph Encryption)** *A graph encryption scheme for distance queries  $\text{Graph} = (\text{Setup}, \text{distQuery})$  consists of a polynomial-time algorithm and a polynomial-time two-party protocol that work as follows:*

- $(K, \text{EGR}) \leftarrow \text{Setup}(1^k, G, \alpha, \varepsilon)$ : *is a probabilistic algorithm that takes as input a security parameter  $k$ , a graph  $G$ , an approximation factor  $\alpha$ , and an error parameter  $\varepsilon$ . It outputs a secret key  $K$  and an encrypted graph  $\text{EGR}$ .*
- $(d, \perp) \leftarrow \text{distQuery}_{C,S}((K, q), \text{EGR})$ : *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  $\text{EGR}$ . 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.*

We say that  $\text{Graph}$  is  $(\alpha, \varepsilon)$ -correct if for all  $k \in \mathbb{N}$ , for all  $G$ , for all  $\alpha \geq 1$ , for all  $\varepsilon < 1$ , and for all  $q = (u, v) \in V^2$ ,

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

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

#### 3.1 Security and Leakage

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

Such a security notion can be difficult to achieve efficiently, so often one allows for some form of leakage. Following [13, 6], 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 distQuery protocol.

We adapt the notion of adaptive semantic security from [13, 6] to our setting (i.e., to the case of graph encryption with support for approximate shortest distance queries).

**Definition 3.2 (Adaptive semantic security)** *Let  $\text{Graph} = (\text{Setup}, \text{distQuery})$  be a graph encryption scheme that supports approximate shortest distance queries 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  $\mathcal{L}_{\text{Setup}}$  and  $\mathcal{L}_{\text{Query}}$  are (stateful) leakage functions:  $\text{Ideal}_{\mathcal{A},\mathcal{S}}(1^k)$ :*

- $\mathcal{A}$  outputs a graph  $G = (V, E)$ , an approximation factor  $\alpha$  and an error parameter  $\varepsilon$ .
- Given  $\mathcal{L}_{\text{Setup}}(G), 1^k, \alpha$  and  $\varepsilon$ ,  $\mathcal{S}$  generates and sends an encrypted graph EGR to  $\mathcal{A}$ .
- $\mathcal{A}$  generates a polynomial number of adaptively chosen queries  $(q_1, \dots, q_m)$ . For each  $q_i$ ,  $\mathcal{S}$  is given  $\mathcal{L}_{\text{Query}}(G, q_i)$  and  $\mathcal{A}$  and  $\mathcal{S}$  execute a simulation of 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.

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

- $\mathcal{A}$  outputs a graph  $G = (V, E)$ , an approximation factor  $\alpha$  and an error parameter  $\varepsilon$ .
- $\mathcal{C}$  computes  $(K, \text{EGR}) \leftarrow \text{Setup}(1^k, G, \varepsilon)$  and sends the encrypted graph EGR to  $\mathcal{A}$ .
- $\mathcal{A}$  generates a polynomial number of adaptively chosen queries  $(q_1, \dots, q_m)$ . For each query  $q_i$ ,  $\mathcal{A}$  and  $\mathcal{C}$  execute  $\text{distQuery}_{\mathcal{C},\mathcal{A}}((K, q), \text{EGR})$ .
- $\mathcal{A}$  computes a bit  $b$  that is output by the experiment.

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

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

In the definition above, it captures the fact that even if the adversarial server choose a graph database on his own, the server still cannot learn any other information just from the encryption scheme by just looking at the encrypted graph and token. All the graph encryption schemes we discuss in this work leak information about the queries. In particular, our first two constructions reveal to the server whether the nodes in a shortest distance query have occurred in a previous query. We formalize this leakage below.

**Definition 3.3 (Query pattern)** *For two queries  $q, q'$  define  $\text{Sim}(q, q') = (u = u', u = v', v = u', v = v')$ , i.e., the equality predicate of whether two queries are equal. Let  $\mathbf{q} = (q_1, \dots, q_m)$  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  $\text{Sim}(q_i, q_j)$ . Note that  $\mathcal{L}_{QP}$  does not leak the identities of the queried nodes.*

Our third construction leaks both the query pattern and leakage we refer to as *sketch pattern*, which we describe in Section 4.

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 3.2 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 [16, 21] for a discussion of privacy of approximation algorithms).

### 3.2 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

## 4 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  $\text{Sk}_v$  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  $\text{Sk}_u$  and  $\text{Sk}_v$ . The first construction is by Das Sarma *et al.* [14] which is itself based on a construction of Thorup and Zwick [44] and the second is by Cohen *et al.* [10]. The two solutions produce sketches of the same form and distance queries are answered using the same operation.

**Sketched-based oracles** More formally, a sketch-based distance oracle  $\text{DO} = (\text{Setup}, \text{Query})$  is a pair of efficient algorithms that work as follows. **Setup** takes as input a graph  $G$ , an approximation factor  $\alpha$  and an error bound  $\varepsilon$  and outputs an oracle  $\Omega_G = \{\text{Sk}_v\}_{v \in V}$ . **Query** takes as input an oracle  $\Omega_G$  and a shortest distance query  $q = (u, v)$ . We say that  $\text{DO}$  is  $(\alpha, \varepsilon)$ -correct if for all graphs  $G$  and all queries  $q = (u, v)$ ,

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

where  $d := \text{Query}(\Omega_G, u, v)$ .

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

**The Cohen *et al.* oracle** The Setup algorithm assigns to each node  $v \in V$ , a sketch  $\text{Sk}_v$  that includes pairs  $(w, \delta)$  chosen as follows. It first chooses a random rank function  $\text{rk} : V \rightarrow [0, 1]$ ; 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  $\rho$  as the oracle’s *rank parameter* and note that it affects the size of the sketches. For each node  $v \in V$ , the sketch  $\text{Sk}_v$  includes pairs  $(w, \delta)$  such that  $\text{rk}(w)$  is less than the  $\rho^{\text{th}}$  value in the sorted set  $\{\text{rk}(y) : y \in N_{\text{dist}(u,v)}(v)\}$ . Finally it outputs a distance oracle  $\Omega_G = (\text{Sk}_{v_1}, \dots, \text{Sk}_{v_n})$ .

**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  $\text{Sk}_u$  and  $\text{Sk}_v$  and returns the minimum over  $s \in \mathbf{I}$  of  $\text{dist}(u, s) + \text{dist}(s, v)$ . If there are no nodes in common, then it returns  $\perp$ .

$\text{Sk}(v_i): \{(a, 3), (b, 3), (e, 6), (g, 3), (h, 4)\}$
$\text{Sk}(v_j): \{(b, 2), (d, 1), (e, 3), (h, 3), (f, 7)\}$

Figure 1: Two example sketches for nodes  $v_i$  and  $v_j$ . The approximate shortest distance  $d = 5$ , since  $b$  is in both sketches and the sum of its distances to  $v_i$  and  $v_j$  is the minimum sum.

**Sketch leakage** We described in Definition 3.3 the query pattern leakage which formally captures the leakage of our first two constructions. The leakage revealed by our third construction (see Section 5.3) is more complex, however. It includes both the query pattern leakage and what we refer to as the sketch pattern leakage which we formalize here.

**Definition 4.1 (Sketch pattern leakage)** *The sketch pattern leakage function  $\mathcal{L}_{SP}(G, q)$  for a graph  $G$  and a query  $q = (u, v)$  is a pair  $(X, Y)$ , where  $X = \{f(w) : (w, \delta) \in \text{Sk}_u\}$  and  $Y = \{f(w) : (w, \delta) \in \text{Sk}_v\}$  are multi-sets and  $f$  is a random function.*

## 5 Our Constructions

### 5.1 A Computationally-Efficient Scheme

We now describe our first scheme which is quite practical. The scheme, described below, makes use of symmetric-key primitives which results in a simple and very efficient construction. The scheme  $\text{GraphEnc}_1 = (\text{Setup}, \text{distQuery})$  makes use of a symmetric-key encryption scheme  $\text{SKE} = (\text{Gen}, \text{Enc}, \text{Dec})$ , a pseudo-random permutation  $P$  and a sketched-based distance oracle  $\text{DO} = (\text{Setup}, \text{Query})$ .

The **Setup** algorithm works as follows. Given a  $1^k, G, \alpha$  and  $\varepsilon$  as inputs:

- It first computes a distance oracle  $\Omega_G$  using  $\text{DO.Setup}(G, \alpha, \varepsilon)$ . It then pads each sketch to be the maximum sketch size  $S_m$  by filling them with dummy values.
- It then generates keys  $K_1, K_2$  for the encryption scheme and pseudo-random permutation respectively and sets  $K = (K_1, K_2)$ . For all nodes  $v \in V$ , it computes a label  $P_{K_2}(v)$  and creates an encrypted sketch  $\text{ESk}_v = (c_1, \dots, c_\lambda)$ , where  $c_i \leftarrow \text{Enc}_{K_1}(w_i \parallel \delta_i)$  is a symmetric-key encryption of the  $i$ th pair  $(w_i, \delta_i)$  in  $\text{Sk}_v$ .
- It then sets up a dictionary  $\text{DX}$  in which it stores, for all  $v \in V$ , the pairs  $(P_{K_2}(v), \text{ESk}_v)$ , ordered by the labels. The encrypted graph is then simply  $\text{EGR} = \text{DX}$ .

The **distQuery** protocol works as follows. To query  $\text{EGR}$  on  $q = (u, v)$ , the client sends a token  $\text{tk} = (\text{tk}_1, \text{tk}_2) = (P_{K_2}(u), P_{K_2}(v))$  to the server who returns the pair  $\text{ESk}_u := \text{DX}[\text{tk}_1]$  and  $\text{ESk}_v := \text{DX}[\text{tk}_2]$ . The client then decrypts each encrypted sketch and computes  $\min_{s \in \mathbf{I}} \text{dist}(u, s) + \text{dist}(s, v)$  (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  $n$  and  $\sigma$  ( $\rho$  in the case of Cohen *et al.* oracle). We defer a formal proof to the full version of this work.

The communication complexity of the **distQuery** protocol is linear in  $S_m$ , where  $S_m$  is the maximum sketch size. Note that even though  $S_m$  is sub-linear in  $n$ , it could still be large in practice. For example, in the Das Sarma *et al.* construction  $S_m = O(n^{2/\alpha} \cdot \log n)$ .

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  $\text{GraphEnc}_2 = (\text{Setup}, \text{distQuery})$  which is less efficient computationally 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  $\text{SWHE} = (\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval})$ , a pseudo-random permutation  $P$ , a family of universal hash functions  $\mathcal{H}$  and a sketch-based distance oracle  $\text{DO} = (\text{Setup}, \text{Query})$ .

The **Setup** algorithm works as follows. Given  $1^k, G, \alpha$  and  $\varepsilon$  as inputs, it generates a public/secret-key pair  $(\text{pk}, \text{sk})$  for SWHE. It then constructs a distance oracle  $\Omega_G$  using  $\text{DO.Setup}(G, \alpha, \varepsilon)$ . Let  $D_m$  be the maximum distance over all the sketches and  $S_m$  be the

---

**Algorithm 1:** Setup algorithm for GraphEnc<sub>2</sub>


---

**Input** :  $1^k, G, \alpha, \varepsilon$   
**Output:** EGR  
**1 begin** Setup  
**2**   Sample  $K \xleftarrow{\$} \{0, 1\}^k$ ;  
**3**   Initialize a dictionary DX;  
**4**   Generate a key pair  $(\text{pk}, \text{sk}) \leftarrow \text{SWHE.Gen}(1^k)$ ;  
**5**   Compute  $\Omega_G \leftarrow \text{DO.Setup}(G, \alpha, \varepsilon)$ ;  
**6**   Set  $S_m := \max_{v \in V} |\text{Sk}_v|$ ;  
**7**   Set  $D_m := \max_{v \in V} \{ \max_{(w, \delta) \in \text{Sk}_v} \delta \}$ ;  
**8**   Set  $N := 2 \cdot D_m + 1$  and  $t = 2 \cdot S_m^2 \cdot \varepsilon^{-1}$ ;  
**9**   Sample a hash function  $h : V \rightarrow [t]$  from  $\mathcal{H}$ ;  
**10**   **foreach**  $v \in V$  **do**  
**11**     compute  $\ell_v := P_K(v)$ ;  
**12**     initialize an array  $\mathbb{T}_v$  of size  $t$ ;  
**13**     **foreach**  $(w_i, \delta_i) \in \text{Sk}_v$  **do**  
**14**       set  $\mathbb{T}_v[h(w_i)] \leftarrow \text{SWHE.Enc}_{\text{pk}}(2^{N-\delta_i})$ ;  
**15**       fill remaining cells of  $\mathbb{T}_v$  with encryptions of 0; set  $\text{DX}[\ell_v] := \mathbb{T}_v$ ;  
**16**     **end**  
**17**   **end**  
**18**   Output  $K$  and  $\text{EGR} = \text{DX}$   
**19 end**

---

maximum sketch size. Setup sets  $N := 2 \cdot D_m + 1$  and samples a hash function  $h \xleftarrow{\$} \mathcal{H}$  with domain  $V$  and co-domain  $[t]$ , where  $t = 2 \cdot S_m^2 \cdot \varepsilon^{-1}$ .

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

Fig. 2 below provides an example of one of the hash tables  $\mathbb{T}_v$  generated from a sketch  $\text{Sk}_v = \{(w_1, \delta_1), \dots, (w_s, \delta_s)\}$ , where  $s$  is the size of the sketch. For all  $i \in [s]$ , the ciphertext  $\text{Enc}_{\text{pk}}(2^{N-\delta_i})$  is stored at location  $h(w_i)$  of the table  $\mathbb{T}_v$ . For example, we place  $\text{Enc}_{\text{pk}}(2^{2-\delta_j})$  to  $\mathbb{T}_v[h(w_j)]$  since  $h(w_j) = 1$ . Finally, all remaining locations of  $\mathbb{T}_v$  are filled with SWHE encryptions of 0. Notice that, since we are using probabilistic encryption, the encryptions of 0 are different.

	0	$h(w_i)$	2	...	$h(w_j)$	$h(w_k)$	$t-1$
$\mathbb{T}_v$	$\text{Enc}_{\text{pk}}(0)$	$\text{Enc}_{\text{pk}}(2^{N-\delta_i})$	$\text{Enc}_{\text{pk}}(0)$	...	$\text{Enc}_{\text{pk}}(2^{N-\delta_j})$	$\text{Enc}_{\text{pk}}(2^{N-\delta_k})$	$\text{Enc}_{\text{pk}}(0)$

Figure 2: One node's encrypted hash table.

The distQuery protocol works as follows. Given a query  $q = (u, v)$ , the client sends tokens  $(\text{tk}_1, \text{tk}_2) = (P_K(u), P_K(v))$  to the server who uses them to retrieve the hash tables of nodes  $u$  and  $v$  by computing  $\mathbb{T}_u := \text{DX}[\text{tk}_1]$  and  $\mathbb{T}_v := \text{DX}[\text{tk}_2]$ . The server then homomorphically evaluates an inner product over the hash tables. More precisely, it computes  $c := \sum_{i=1}^t \mathbb{T}_u[i] \cdot$

$T_v[i]$ , where  $\sum$  and  $\cdot$  refer to the homomorphic addition and multiplication operations of the SWHE scheme. Finally, the server returns only  $c$  to the client who decrypts it and outputs  $2N - \log_2(\text{Dec}_{\text{sk}}(c))$ .

---

**Algorithm 2:** DistQuery algorithm for GraphEnc<sub>2</sub>

---

**Input** : Client’s input is  $(K, q)$  and server’s input is EGR.  
**Output:** Client’s output is  $\text{dist}_q$  and server’s output is  $\perp$ .

```

1 begin distQuery
2    $C$ : client parses  $q$  as  $(u, v)$ ;
3    $C \Rightarrow S$ : client sends  $\text{tk} = (\text{tk}_1, \text{tk}_2) = (P_K(u), P_K(v))$ ;
4    $S$ : server retrieves  $T_1 := \text{DX}[\text{tk}_1]$  and  $T_2 := \text{DX}[\text{tk}_2]$ ;
5   foreach  $i \in [t]$  do
6     | Server computes  $c_i \leftarrow \text{SWHE.Eval}(\times, T_1[i], T_2[i])$ ;
7   end
8    $S \Rightarrow C$ : server sends  $c \leftarrow \text{SWHE.Eval}(+, c_1, \dots, c_t)$ ;
9    $C$ : client computes  $m \leftarrow \text{SWHE.Dec}_{\text{sk}}(c)$ ;
10   $C$ : client outputs  $\text{dist} = 2N - \log m$ .
11 end

```

---

Note that the storage complexity at the server is  $O(n \cdot t)$  and the communication complexity of `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  $\delta_i$  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.

**Instantiating and optimizing the SWHE** For our experiments (see Section 6) we instantiate the SWHE scheme with the BGN construction of [2]. We chose BGN due to the efficiency of its encryption algorithm and the compactness of its ciphertexts and keys. 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 [39] 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 and Security

Here, we analyze the correctness of GraphEnc<sub>2</sub>. We first bound the collision probability of our construction and then proceed to prove correctness of our construction 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 collisions occurred in the Setup algorithm while constructing the hash tables  $T_u$  and  $T_v$ . Then,  $\Pr[\mathcal{E}_q] \leq 2 \cdot \frac{S_m^2}{t}$ .*

**Proof:** Let  $\text{Coll}_v$  be the event that at least one collision occurs while creating  $v$ 's hash table  $\mathsf{T}_v$  (i.e., in Algorithm 1 Setup Line 14). Also, let  $\text{XColl}_{u,v}$  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 query  $q = (u, v)$ , we have

$$\Pr[\mathcal{E}_q] \leq \Pr[\text{Coll}_u] + \Pr[\text{Coll}_v] + \Pr[\text{XColl}_{u,v}]. \quad (1)$$

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

Note that in practice “intra-sketch” collision events  $\text{Coll}_u$  and  $\text{Coll}_v$  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  $\text{XColl}_{u,v}$ , 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  $u \neq v$ ,*

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

where  $(d, \perp) := \text{GraphEnc}_2.\text{distQuery}((K, q), \text{EGR})$  and  $(K, \text{EGR}) \leftarrow \text{GraphEnc}_2.\text{Setup}(1^k, G, \alpha, \varepsilon)$ .

**Proof:** Let  $\mathbf{I}$  be the set of nodes in common between  $\mathsf{Sk}_u$  and  $\mathsf{Sk}_v$  and let  $\text{mindist} = \min_{w_i \in \mathbf{I}} \{\delta_i^u + \delta_i^v\}$ , where for all  $0 \leq i \leq |I|$ ,  $\delta_i^u \in \mathsf{Sk}_u$  and  $\delta_i^v \in \mathsf{Sk}_v$ . Note that at line 8 in Algorithm 2  $\text{distQuery}$ , 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  $\mathsf{T}_u$  and  $\mathsf{T}_v$  of  $u$  and  $v$  respectively. Conditioned on  $\overline{\mathcal{E}}_q$ , we therefore have that

$$\begin{aligned} c &= \sum_{i=1}^{|\mathbf{I}|} \text{Enc}_{\text{pk}}(2^{N-\delta_i^u}) \cdot \text{Enc}_{\text{pk}}(2^{N-\delta_i^v}) \\ &= \text{Enc}_{\text{pk}}\left(2^{2N} \cdot \sum_{i=1}^{|\mathbf{I}|} 2^{-(\delta_i^u + \delta_i^v)}\right), \end{aligned}$$

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

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

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

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

where the first inequality also holds from  $\text{mindist} \leq (\delta_i^u + \delta_i^v)$  for all  $i \in |\mathbf{I}|$ .

Now, by the  $(\alpha, \varepsilon)$ -correctness of DO, we have that  $\text{mindist} \leq \alpha \cdot \text{dist}(u, v)$  with probability at least  $(1 - \varepsilon)$  over the coins of DO.Setup. So, conditioned on  $\overline{\mathcal{E}_q}$ ,

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

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

**Space complexity** Note that to achieve  $(\alpha, \varepsilon)$ -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  $S_m$ , then the size of *every* encrypted sketch is  $t = 2 \cdot S_m^2 \cdot \varepsilon^{-1}$ , 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 negative approximations** Note that in practice, the set of common nodes  $|\mathbf{I}|$  could be large and, in particular, larger than  $\text{mindist}$  which would yield even a negative distance (we indeed observe 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 homomorphic encryptions of the form  $\text{Enc}_{\text{pk}}(2^{N-\delta})$  we could use  $\text{Enc}_{\text{pk}}(B^{N-\delta})$  for  $B = 3$  or  $B = 4$ . This would result in an improved lower bound of  $\text{mindist} - \log_B(|\mathbf{I}|)$  but would also increase the homomorphic decryption time.

**Remark on error rate** Given the above analysis, a client that makes  $\gamma$  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  $O(\gamma \cdot \log^2(n)/\sqrt{n})$  which decreases significantly as  $n$  grows. In the case of 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  $O(\gamma \cdot \ln^2(n)/\sqrt{n})$ . We provide in section 6 detailed experimental result on the error rate.

**Security** In the following Theorem, we analyze the security of GraphEnc<sub>2</sub>.

**Theorem 5.3** *If  $P$  is pseudo-random and SWHE is CPA-secure then GraphEnc<sub>2</sub>, as described above, is adaptively  $(\mathcal{L}_{\text{Setup}}, \mathcal{L}_{\text{Query}})$ -semantically secure, where  $\mathcal{L}_{\text{Setup}}(G) = n$  and  $\mathcal{L}_{\text{Query}}(G, q) = \mathcal{L}_{QP}(q)$ .*

**Proof:** Consider the simulator  $\mathcal{S}$  that works as follows. Given leakage  $\mathcal{L}_{\text{Setup}} = n$ , for all  $1 \leq i \leq n$ , it samples  $\ell_i \xleftarrow{\$} \{0, 1\}^{\log n}$  (without repetition) and creates an array  $\mathbb{T}_i$  of size  $t$  filled with homomorphic encryptions of 0. It then creates a dictionary  $\text{DX}$  and sets  $\text{DX}[\ell_i] = \mathbb{T}_i$  for all  $1 \leq i \leq n$ . Finally, it outputs  $\text{EGR} = \text{DX}$ . Given leakage  $\mathcal{L}_{QP}(q)$ ,  $\mathcal{S}$  first checks whether any of the two query nodes appeared in an earlier query. If the first query node appeared in a previous query,  $\mathcal{S}$  sets  $\text{tk}_1$  to its stored token. Otherwise  $\mathcal{S}$  chooses a fresh token  $\text{tk}_1 \xleftarrow{\$} \{0, 1\}^{\log n}$  and stores it.  $\mathcal{S}$  proceeds similarly with token  $\text{tk}_2$  and then sends  $\text{tk} = (\text{tk}_1, \text{tk}_2)$ .

It now remains to show that the  $\mathbf{Real}_{\mathcal{A}}(1^k)$  and  $\mathbf{Ideal}_{\mathcal{A}, \mathcal{S}}(1^k)$  experiments will output 1 with negligibly-close probability. This can be done using the following sequence of 3 games:

- **Game<sub>0</sub>**: this game corresponds exactly to a  $\mathbf{Real}_{\mathcal{A}}(1^k)$  experiment.
- **Game<sub>1</sub>**: is the same as **Game<sub>0</sub>** except that the output of  $P$  is replaced with random  $(\log n)$ -bit strings. Clearly, the pseudo-randomness of  $P$  guarantees that

$$|\Pr[\text{Game}_0 = 1] - \Pr[\text{Game}_1 = 1]| \leq \text{negl}(k).$$

- **Game<sub>2</sub>**: is the same as **Game<sub>1</sub>** except that all the HE encryptions are replaced with HE encryptions of 0. Clearly, it follows by the CPA-security of SWHE that

$$|\Pr[\text{Game}_1 = 1] - \Pr[\text{Game}_2 = 1]| \leq \text{negl}(k).$$

Note that, by construction, **Game<sub>2</sub>** corresponds exactly to an  $\mathbf{Ideal}_{\mathcal{A}, \mathcal{S}}(1^k)$  experiment so we have

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

from which the Theorem follows. The Theorem follows from the pseudo-randomness of  $P$  and the CPA-security of SWHE.  $\blacksquare$

### 5.3 A Space-Efficient Construction

Although our second construction, **GraphEnc<sub>2</sub>**, achieves optimal communication complexity, it has two limitations. The first is that it is less computationally-efficient than our first construction **GraphEnc<sub>1</sub>** 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 **GraphEnc<sub>1</sub>** by a factor of  $2 \cdot S_m \cdot \epsilon^{-1}$ . 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 **GraphEnc<sub>3</sub>** = (**Setup**, **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.

The details of the scheme are given in Algorithms 3 and 4. At a very high-level, the scheme works similarly to **GraphEnc<sub>2</sub>** 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,

---

**Algorithm 3:** Setup algorithm for GraphEnc<sub>3</sub>


---

```

Input :  $1^k, G, \alpha, \varepsilon$ 
Output: EGR
1 begin Setup
2   Sample  $K_1, K_2 \xleftarrow{\$} \{0, 1\}^k$ ;
3   Initialize a dictionary DX of size  $n$ ;
4   Generate  $(\text{pk}, \text{sk}) \leftarrow \text{SWHE.Gen}(1^k)$ ;
5   Compute  $\Omega_G \leftarrow \text{DO.Setup}(G, \alpha, \varepsilon)$ ;
6   Set  $S_m := \max_{v \in V} |\text{Sk}_v|$ ;
7   Set  $D_m := \max_{v \in V} \{ \max_{(w, \delta) \in \text{Sk}_v} \delta \}$ ;
8   Set  $N := 2 \cdot D_m + 1$  and  $t = 2 \cdot S_m^2 \cdot \varepsilon^{-1}$ ;
9   Initialize collision-resistant hash function  $h$ ;
10  Initialize an array Arr of size  $m = \sum_{v \in V} |\text{Sk}_v|$ ;
11  Sample a random permutation  $\pi$  over  $[m]$ ;
12  foreach  $v \in V$  do
13    sample  $K_v \leftarrow \{0, 1\}^k$ ;
14    foreach  $(w_i, \delta_i) \in \text{Sk}_v$  do
15      compute  $c_i \leftarrow \text{SWHE.Enc}_{\text{pk}}(2^{N-\delta_i})$ ;
16      if  $i \neq |\text{Sk}_v|$  then
17        | Set  $\text{N}_i = \langle h(w_i) \| c_i \| \pi(\text{ctr} + 1) \rangle$ ;
18      else
19        | Set  $\text{N}_i = \langle h(w_i) \| c_i \| \text{NULL} \rangle$ ;
20      end
21      Sample  $r_i \xleftarrow{\$} \{0, 1\}^k$ ;
22      Set  $\text{Arr}[\pi(\text{ctr})] := \langle \text{N}_i \oplus H(K_v \| r_i), r_i \rangle$ ;
23      Set  $\text{ctr} = \text{ctr} + 1$ ;
24    end
25  end
26  foreach  $v \in V$  (in random order) do
27    | Set  $\text{DX}[P_{K_1}(v)] := \langle \text{addr}_{\text{Arr}}(\text{h}_v) \| K_v \rangle \oplus F_{K_2}(v)$ 
28  end
29  Output  $K = (K_1, K_2, \text{pk}, \text{sk})$  and  $\text{EGR} = (\text{DX}, \text{Arr})$ ;
30 end

```

---

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 turns, leaks structural information about the graph.

We overcome this technical difficulty by adapting a similar technique from [13] to our setting. Intuitively, we view the seed/distance pairs in each sketch  $\text{Sk}_v$  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  $\text{SWHE} = (\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$ , a pseudo-random permutation  $P$ , a pseudo-random function  $F$ , a random oracle  $H$ , a collision-resistant hash function  $h$  modeled as a random function and a distance oracle  $\text{DO} = (\text{Setup}, \text{Query})$ .

The **Setup** algorithm takes as input a security parameter  $k$ , a graph  $G$ , an approximation factor  $\alpha > 0$  and an error parameter  $\varepsilon < 1$ . As shown in Algorithm 3, it first constructs a distance oracle  $\Omega_G$  using  $\text{DO.Setup}(G, \alpha, \varepsilon)$ , initializes a counter  $\text{ctr} = 0$  and samples a random permutation  $\pi$  over the domain  $[m]$ , where  $m = \sum_{v \in V} |\text{Sk}_v|$ . It then initializes an  $m$ -size array **Arr**. It then proceeds to create an encrypted sketch  $\text{ESk}_v$  from each sketch  $\text{Sk}_v$  as follows. First samples a symmetric key  $K_v$  for this sketch. Then, for each seed/distance pair  $(w_i, \delta_i)$  in  $\text{Sk}_v$ , it creates a linked-list node  $N_i = \langle h(w_i) \| c_i \| \pi(\text{ctr} + 1) \rangle$ , where  $c_i \leftarrow \text{Enc}_{\text{pk}}(2^{N-\delta_i})$ , and stores an  $H$ -based encryption  $\langle N_i \oplus H(K_v \| r_v), r_v \rangle$  of the node at location  $\pi(\text{ctr})$  in **Arr**. For the last seed/distance pair, it uses instead a linked-list node of the form  $N_i = \langle h(w_i) \| c_i \| \text{NULL} \rangle$ . It then increments  $\text{ctr}$ .

**Setup** then creates a dictionary **DX** where it stores for each node  $v \in V$ , the pair  $(P_{K_1}(v), \langle \text{addr}_{\text{Arr}}(h_v) \| K_v \rangle \oplus F_{K_2}(v))$ , where  $\text{addr}_{\text{Arr}}(h_v)$  is the location in **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  $\text{Sk}_u$  has the element  $(a, d_1), (b, d_2), (c, d_3)$ . The locations  $\text{ind1}, \text{ind2}, \text{ind3}$  in **Arr** are computed according the random permutation  $\pi$ .

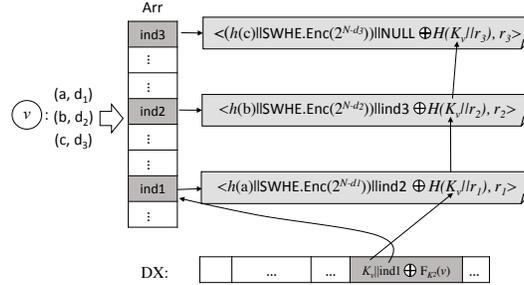


Figure 3: Example of encrypting the sketch  $\text{Sk}_u = \{(a, d_1), (b, d_2), (c, d_3)\}$ .

The **distQuery** protocol, which is shown in Algorithm 4, works as follows. Given a query  $q = (u, v)$ , the client sends tokens  $(\text{tk}_1, \text{tk}_2, \text{tk}_3, \text{tk}_4) = (P_{K_1}(u), P_{K_1}(v), F_{K_2}(u), F_{K_2}(v))$  to the server who uses them to retrieve the values  $\gamma_1 := \text{DX}[\text{tk}_1]$  and  $\gamma_2 := \text{DX}[\text{tk}_2]$ . The server computes  $\langle a_1 \| K_u \rangle := \gamma_1 \oplus \text{tk}_3$  and  $\langle b_1 \| K_v \rangle := \gamma_2 \oplus \text{tk}_4$ .

Next, it recovers the lists pointed to by  $a_1$  and  $b_1$ . More precisely, starting with  $i = 1$ , it parses  $\text{Arr}[a_1]$  as  $\langle \sigma_u, r_u \rangle$  and decrypts  $\sigma_u$  by computing  $\langle h_i \| c_i \| a_{i+1} \rangle := \sigma_u \oplus H(K_u \| r_u)$  while  $a_{i+1} \neq \text{NULL}$ . And starting with  $j = 1$ , it does the same to recover  $\langle h'_j \| c'_j \| b_{j+1} \rangle$  while  $b_{j+1} \neq \text{NULL}$ .

The server then homomorphically computes an inner product over the ciphertext with the same hashes. More precisely, it computes  $\text{ans} := \sum_{(i,j): h_i = h'_j} c_i \cdot c'_j$ , 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 who decrypts it and outputs  $2N - \log_2(\text{SWHE.Dec}_{\text{sk}}(\text{ans}))$ .

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 **GraphEnc<sub>3</sub>** follows directly from the correctness of **GraphEnc<sub>2</sub>**. To see why, observe that: (1) the homomorphic encryptions stored in the encrypted graph of **GraphEnc<sub>3</sub>**

---

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

---

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

```
1 begin distQuery
2    $C$ : computes  $(\text{tk}_1, \text{tk}_2, \text{tk}_3, \text{tk}_4) = (P_{K_1}(u), P_{K_1}(v), F_{K_2}(u), F_{K_2}(v))$ ;
3    $C \Rightarrow S$ : sends  $\text{tk} = (\text{tk}_1, \text{tk}_2, \text{tk}_3, \text{tk}_4)$ ;
4    $S$ : computes  $\gamma_1 \leftarrow \text{DX}[\text{tk}_1]$  and  $\gamma_2 \leftarrow \text{DX}[\text{tk}_2]$ ;
5   if  $\gamma_1 = \perp$  or  $\gamma_2 = \perp$  then
6     | exit and return  $\perp$  to the client
7   end
8    $S$ : compute  $\langle a_1 \| K_u \rangle := \gamma_1 \oplus \text{tk}_3$ ;
9    $S$ : parse  $\text{Arr}[a_1]$  as  $\langle \sigma_u, r_u \rangle$ ;
10   $S$ : compute  $N_1 := \sigma_u \oplus H(K_u \| r_u)$ ;
11  repeat
12    | parse  $N_i$  as  $\langle h_i \| c_i \| a_{i+1} \rangle$ ;
13    | parse  $\text{Arr}[a_{i+1}]$  as  $\langle \sigma_{i+1}, r_{i+1} \rangle$ ;
14    | compute  $N_{i+1} := \sigma_{i+1} \oplus H(K_u \| r_{i+1})$ ;
15    | set  $i = i + 1$ ;
16  until  $a_{i+1} = \text{NULL}$ ;
17   $S$ : compute  $\langle b_1 \| K_v \rangle := \gamma_2 \oplus \text{tk}_4$ ;
18   $S$ : parse  $\text{Arr}[b_1]$  as  $\langle \sigma_v, r_v \rangle$ ;
19   $S$ : compute  $N'_1 := \sigma_v \oplus H(K_v \| r_v)$ ;
20  repeat
21    | parse  $N'_j$  as  $\langle h'_j \| c'_j \| b_{j+1} \rangle$ ;
22    | parse  $\text{Arr}[b_{j+1}]$  as  $\langle \sigma_{j+1}, r_{j+1} \rangle$ ;
23    | compute  $N'_{j+1} := \sigma_{j+1} \oplus H(K_v \| r_{j+1})$ ;
24    | set  $j = j + 1$ ;
25  until  $b_{j+1} = \text{NULL}$ ;
26   $S$ : set  $s := \text{SWHE.Enc}_{\text{pk}}(0)$ ;
27  foreach  $(N_i, N'_j)$  do
28    | if  $h_i = h'_j$  then
29      | compute  $p := \text{SWHE.Eval}(\times, c_i, c'_j)$ ;
30      | compute  $s := \text{SWHE.Eval}(+, s, p)$ ;
31    | end
32  end
33   $S \Rightarrow C$ : send  $s$ ;
34   $C$ : compute  $d := \text{SWHE.Dec}_{\text{sk}}(s)$ 
35 end
```

---

are the same as those in the encrypted graph produced by  $\text{GraphEnc}_2$  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  $\text{GraphEnc}_2$ , with the exception of the homomorphic sums with 0-encryptions.

We note that  $\text{GraphEnc}_3$  leaks only a little more than the previous constructions. Intuitively, for a query  $q = (u, v)$ , the leakage consists of revealing to the server: (1) which seed/distance pairs in the sketches  $\text{Sk}_u$  and  $\text{Sk}_v$  are the same; and (2) the size of these sketches. This is formalized in Definition 4.1 as the sketch pattern leakage  $\mathcal{L}_{SP}(G, q)$ . In the following Theorem, we summarize the security of  $\text{GraphEnc}_3$ .

**Theorem 5.4** *If  $P$  and  $F$  are pseudo-random, if SWHE is CPA-secure then GraphEnc<sub>3</sub>, as described above, is adaptively  $(\mathcal{L}_{\text{Setup}}, \mathcal{L}_{\text{Query}})$ -semantically secure in the random oracle model, where  $\mathcal{L}_{\text{Setup}}(G) = (n, m)$  and  $\mathcal{L}_{\text{Query}}(G, q) = (\mathcal{L}_{QP}(G, q), \mathcal{L}_{SP}(G, q))$ .*

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

Given leakage  $\mathcal{L}_{\text{Query}}(G, q) = (\mathcal{L}_{QP}(G, q), \mathcal{L}_{SP}(G, q))$  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  $\text{tk}_1$  and  $\text{tk}_3$  to the values that were previously used. If not, it sets  $\text{tk}_1 := \ell_i$  for some previously unused  $\ell_i$  and  $\text{tk}_3$  as follows. It chooses a previously unused  $\alpha \in [m]$  at random, a key  $K_u \xleftarrow{\$} \{0, 1\}^k$  and sets  $\text{tk}_3 := \text{DX}[\text{tk}_1] \oplus \langle \alpha \| K_u \rangle$ . It then remembers the association between  $K_u$  and  $X$  and the sketch size  $|\text{Sk}_u|$ . It does the same for the query node  $v$ , sets  $\text{tk}_2$  and  $\text{tk}_4$  analogously and associates  $|\text{Sk}_v|$  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  $\mathbf{Arr}$  has the form  $\langle s, r \rangle$  where  $s$  is a  $(\log t + g(N) + \log m)$ -bit string. If  $K$  has not been queried before, it initializes a counter  $\text{ctr}_K := 0$ . If an appropriate entry exists in  $\mathbf{Arr}$ , it returns  $s \oplus \langle \gamma, c, p \rangle$ , where  $\gamma$  is the  $\text{ctr}^{\text{th}}$  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  $\mathbf{Arr}$  chosen at random or  $\emptyset$  if  $\text{ctr} = |\text{Sk}|$ , where  $|\text{Sk}|$  is the sketch size associated with  $K$ . If no appropriate entry exists in  $\mathbf{Arr}$ ,  $\mathcal{S}$  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.* and Cohen *et al.* distance oracles and all three of our graph encryption schemes.

We use the AES-128 in CBC mode for symmetric encryption and instantiate SWHE with the Boneh-Goh-Nissim (BGN) scheme, implement in C++ with the Stanford Pairing-Based Library PBC<sup>1</sup>. We use the standard openssl<sup>2</sup> for all basic cryptographic tools and use 128-bit security for all the encryptions. We use HMAC for PRFs and instantiate the hash function in GraphEnc<sub>3</sub> with HMAC-SHA-256. All experiment were run on a 24-core 2.9GHz Intel Xeon, with 512 GBs of RAM running Linux.

**Datasets** We use real-world graph datasets publicly available from the Stanford SNAP website<sup>3</sup>. 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

<sup>1</sup><http://crypto.stanford.edu/xbc/>

<sup>2</sup><https://www.openssl.org/>

<sup>3</sup><https://snap.stanford.edu/data/>

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
as-skitter	1,696,415	11,095,298	25
com-Youtube	1,134,890	2,987,624	20
loc-Gowalla	196,591	950,327	14
email-Enron	36,692	367,662	11
ca-CondMat	23,133	186,936	14

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 [28] and is true for expander and small-world graphs, which is known to model many real-life graphs. The implication of this, is that the maximum distance  $D_m$  in the sketches generated by the distance oracles is, in practice, small and therefore the value  $N$  that we use in GraphEnc<sub>2</sub> and GraphEnc<sub>3</sub> (see Algorithm 1 and 3) is typically small.

**Performance** Table 2 below gives our constructions’ space, setup and communication complexities and Table 3 summarizes our experimental results. (Note that  $S_m$  is mainly based the parameters of the distance oracle ( $\sigma$  in  $\mathbf{DO}_1$ ,  $\rho$  in  $\mathbf{DO}_2$ ) and we could achieve very good approximation for  $S_m = O(\log n)$  see [14, 10]).

Scheme	Space	Setup	Comm.
GraphEnc <sub>1</sub>	$O(nS_m)$	$O(nS_m)$	$O(S_m)$
GraphEnc <sub>2</sub>	$O(nt)$	$O(nt)$	$O(1)$
GraphEnc <sub>3</sub>	$O(nS_m)$	$O(nS_m)$	$O(1)$

Table 2: The space, setup and communication complexities of our constructions.

## 6.1 Performance of GraphEnc<sub>1</sub>

Dataset	sketch size $S_m$	Graph Sketching Scheme	GraphEnc <sub>1</sub>			GraphEnc <sub>2</sub>				GraphEnc <sub>3</sub>		
			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 GraphEnc<sub>1</sub>, GraphEnc<sub>2</sub>, and GraphEnc<sub>3</sub>

**Distance oracle parameters** In the following Sections we evaluate the performance of our constructions 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.,  $S_m$ ) of roughly the same amount.

We can see from Table 3 that the time to setup an encrypted graph with  $\mathbf{GraphEnc}_1$  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  $\mathbf{GraphEnc}_1.\mathbf{Setup}$  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 test for various sketch sizes, and the communication per query went up to 3.8KB for *as-skitter* when we set  $S_m = 80$ . This can become quite significant if the server is interacting with multiple clients.

## 6.2 Performance of $\mathbf{GraphEnc}_2$

The first column in Table 3 of the  $\mathbf{GraphEnc}_2$  experiments gives the size the encrypted hash tables  $T_v$  constructed during  $\mathbf{GraphEnc}_2.\mathbf{Setup}$ . 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  $\mathbf{GraphEnc}_2.\mathbf{Setup}$  be quite costly, about 3 orders of magnitude more expensive than  $\mathbf{GraphEnc}_1.\mathbf{Setup}$ . This is mostly due to generating the 0-encryptions. Note, however, that similarly to  $\mathbf{GraphEnc}_1$ , we can use extensive parallelization to speed up the graph encryption. 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,  $\mathbf{GraphEnc}_2$  achieves a constant communication cost of 34B.

**Collisions** 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 say from the results, the oracle  $\mathbf{DO}_2$  has less error rate than  $\mathbf{DO}_1$ . We would like to point out that these 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.

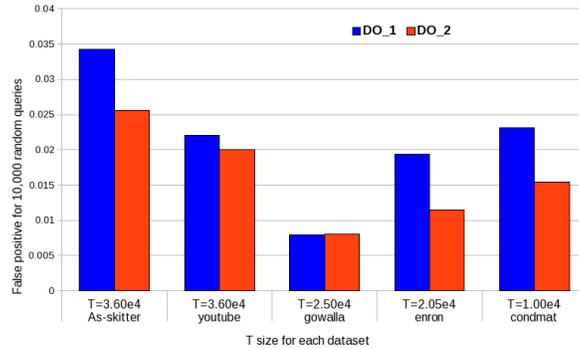


Figure 4: Collision probabilities for different datasets

### 6.3 Performance of GraphEnc<sub>3</sub>

The GraphEnc<sub>3</sub> columns in Table 3 show that GraphEnc<sub>3</sub> is as efficient as GraphEnc<sub>1</sub> in terms of setup time and encrypted sketch size. Moreover, it achieves  $O(1)$  communication of 34B like GraphEnc<sub>2</sub>. Using a single machine, GraphEnc<sub>3</sub>.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 instantiate the hash function  $h$  using a cryptographic keyed hash function HMAC-SHA-256 and this worked very well in all our experiments.

**Construction time & encrypted sketch size** Since the performance of GraphEnc<sub>3</sub> depends only on the size of the underlying sketches we investigate the relationship between the performance of GraphEnc<sub>3</sub>.Setup and the sampling and rank parameters of the Das Sarma *et al.* and Cohen *et al.* oracles, respectively. We use values of  $\sigma$  and  $\rho$  ranging from 3 to 6 in each case which resulted in maximum sketch sizes  $S_m$  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  $\sigma$  and  $\rho$  increase. In fact, the results show that GraphEnc<sub>3</sub> does not add much overhead when using more seed/distance pairs in the sketches. Also, unlike the previous schemes, GraphEnc<sub>3</sub> produces encrypted sketches that are compact since it does not use 0-encryptions for padding purposes.

**Query time** We measure 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 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 more a careful implementation (e.g., faster pairing library) would reduce

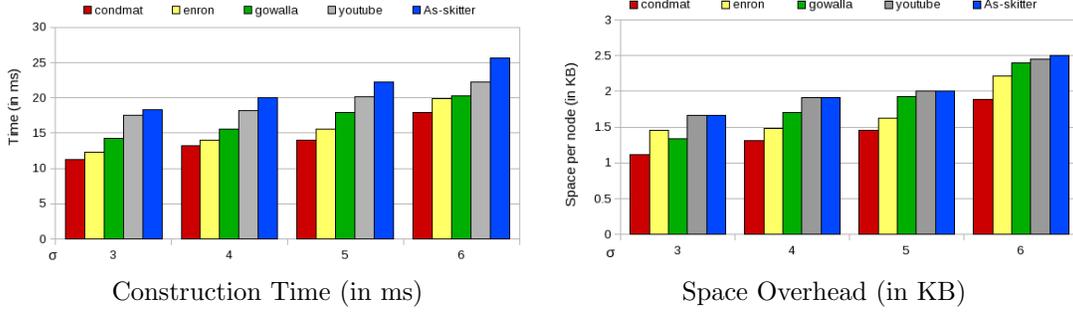


Figure 5: Construction time and size overhead ( $\mathbf{DO}_1$ )

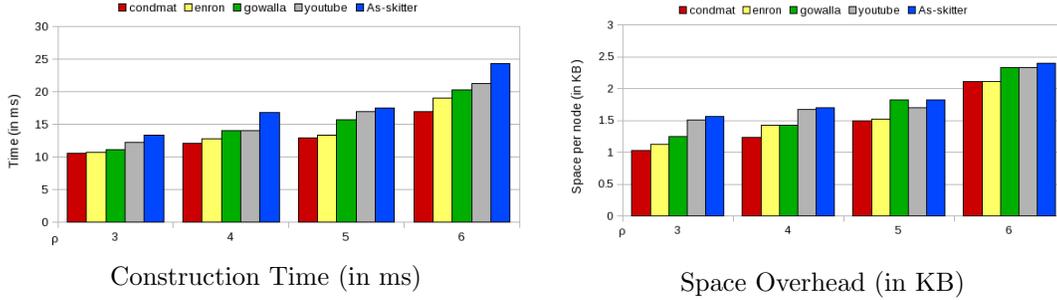
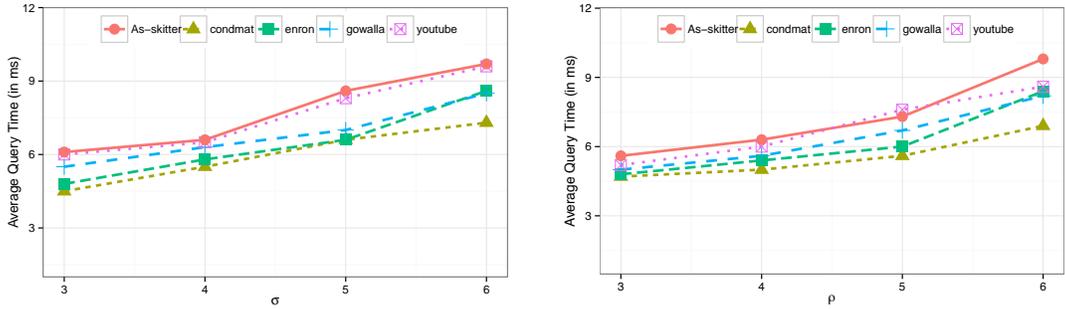


Figure 6: Construction time and size overhead ( $\mathbf{DO}_2$ )



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

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

Figure 7: Average Query time

the query time even further.

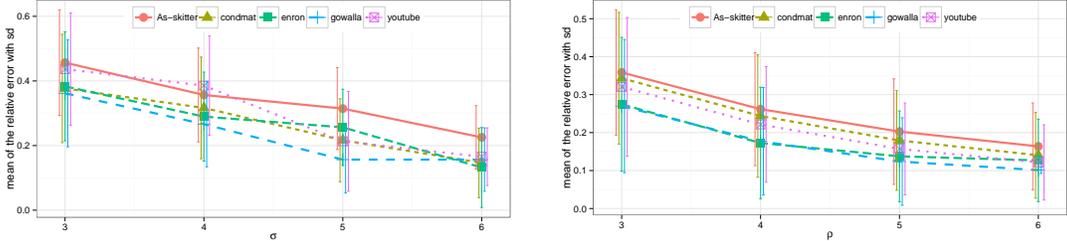
We also measure the decryption time at the client. As pointed out previously, decryption time depends on  $N$  which itself is a function of the diameter of the graph. Since all our graphs have small diameter, client decryption time—which itself consists of a BGN decryption—was performed very efficiently. 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 through our implementation. We leak the parameter  $\rho$  and  $\sigma$  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.

### 6.4 Approximation errors

We investigate the approximation errors produced by our schemes. We generate 10K random queries and run the  $\text{Query}_{C,S}$  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.



Mean of Estimated Error with Standard Deviation using  $\text{DO}_1$

Mean of Estimated Error with Standard Deviation using  $\text{DO}_2$

Figure 8: Mean of Estimated Error

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. For the *Gowalla*, the mean of the relative error ranges from 0.36 to 0.13 when using the Das Sarma *et al.* oracle  $\text{DO}_1$ . 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  $\text{DO}_2$  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  $\text{DO}_2$  with  $\rho = 3$ . All the other datasets are very similar to them, we omit them due to space limit.

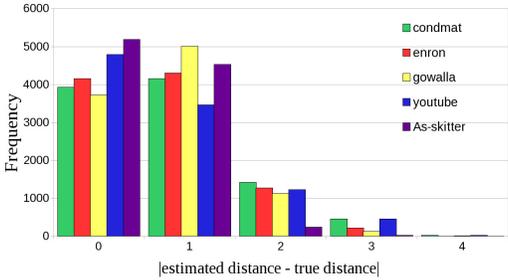


Figure 9: Absolute error histogram  $\text{DO}_2$  and  $\rho = 3$

We note that a very small number of distances were negative and we remove 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  $d \geq \text{mindist} - \log |\mathbf{I}|$ . If  $|\mathbf{I}|$  is large and  $\text{mindist}$  is small (say, 1 or 2) then it is very likely that  $d$  is negative.

## 7 Conclusion

In this work, we describe three graph encryption schemes that support approximate shortest distance queries. Our first solution, **GraphEnc<sub>1</sub>**, is based only on symmetric-key primitives and is computationally very efficient while our second solution, **GraphEnc<sub>2</sub>**, is based on somewhat homomorphic encryption and is optimal in terms of communication complexity. Furthermore, our third solution, **GraphEnc<sub>3</sub>**, 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 implement our constructions and evaluate their efficiency experimentally, showing that our constructions are practical for large-scale graphs.

## References

- [1] BLANTON, M., STEELE, A., AND ALIASGARI, M. Data-oblivious graph algorithms for secure computation and outsourcing. In *ASIACCS* (2013), pp. 207–218.
- [2] BONEH, D., GOH, E.-J., AND NISSIM, K. Evaluating 2-dnf formulas on ciphertexts. In *TCC 2005* (2005), pp. 325–342.
- [3] CASH, D., JAEGER, J., JARECKI, S., JUTLA, C., KRAWCZYK, H., ROSU, M., AND STEINER, M. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS '14)* (2014).
- [4] CASH, D., JARECKI, S., JUTLA, C., KRAWCZYK, H., ROSU, M., AND STEINER, M. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO '13* (2013), pp. 353–373.
- [5] CHANG, Y., AND MITZENMACHER, M. Privacy preserving keyword searches on remote encrypted data. In *ACNS '05* (2005), Springer, pp. 442–455.
- [6] CHASE, M., AND KAMARA, S. Structured encryption and controlled disclosure. In *ASIACRYPT '10* (2010), vol. 6477, pp. 577–594.
- [7] CHECHIK, S. Approximate distance oracles with constant query time. In *STOC* (2014), pp. 654–663.
- [8] CHENG, J., FU, A. W.-C., AND LIU, J. K-isomorphism: privacy preserving network publication against structural attacks. In *SIGMOD Conference* (2010), pp. 459–470.
- [9] COHEN, E. All-distances sketches, revisited: Hip estimators for massive graphs analysis. In *PODS* (2014), pp. 88–99.
- [10] COHEN, E., DELLING, D., FUCHS, F., GOLDBERG, A. V., GOLDSZMIDT, M., AND WERNECK, R. F. Scalable similarity estimation in social networks: closeness, node labels, and random edge lengths. In *COSN* (2013), pp. 131–142.
- [11] COHEN, E., HALPERIN, E., KAPLAN, H., AND ZWICK, U. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.* 32, 5 (2003), 1338–1355.
- [12] CORMODE, G., SRIVASTAVA, D., YU, T., AND ZHANG, Q. Anonymizing bipartite graph data using safe groupings. *PVLDB* (2008), 833–844.

- [13] CURTMOLA, R., GARAY, J., KAMARA, S., AND OSTROVSKY, R. Searchable symmetric encryption: Improved definitions and efficient constructions. In *CCS 2006* (2006), ACM, pp. 79–88.
- [14] DAS SARMA, A., GOLLAPUDI, S., NAJORK, M., AND PANIGRAHY, R. A sketch-based distance oracle for web-scale graphs. In *WSDM* (2010), pp. 401–410.
- [15] DWORK, C., MCSHERRY, F., NISSIM, K., AND SMITH, A. Calibrating noise to sensitivity in private data analysis. In *TCC* (2006), pp. 265–284.
- [16] FEIGENBAUM, J., ISHAI, Y., MALKIN, T., NISSIM, K., STRAUSS, M. J., AND WRIGHT, R. N. Secure multiparty computation of approximations. *ACM Transactions on Algorithms* 2, 3 (2006), 435–472.
- [17] GAO, J., YU, J. X., JIN, R., ZHOU, J., WANG, T., AND YANG, D. Neighborhood-privacy protected shortest distance computing in cloud. In *SIGMOD* (2011), pp. 409–420.
- [18] GENTRY, C. Fully homomorphic encryption using ideal lattices. In *STOC '09* (2009), ACM Press, pp. 169–178.
- [19] GENTRY, C., HALEVI, S., AND VAIKUNTANATHAN, V. A simple bgn-type cryptosystem from lwe. In *Advances in Cryptology - EUROCRYPT '10* (2010), Springer, pp. 506–522.
- [20] GOLDBREICH, O., AND OSTROVSKY, R. Software protection and simulation on oblivious RAMs. *Journal of the ACM* 43, 3 (1996), 431–473.
- [21] HALEVI, S., KRAUTHGAMER, R., KUSHILEVITZ, E., AND NISSIM, K. Private approximation of np-hard functions. In *STOC* (2001), ACM, pp. 550–559.
- [22] HAN, W., LEE, S., PARK, K., LEE, J., KIM, M., KIM, J., AND YU, H. Turbograph: a fast parallel graph engine handling billion-scale graphs in a single PC. In *KDD* (2013), pp. 77–85.
- [23] KAMARA, S., AND PAPAMANTHOU, C. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security (FC '13)* (2013).
- [24] KAMARA, S., PAPAMANTHOU, C., AND ROEDER, T. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security (CCS '12)* (2012), ACM Press.
- [25] KASIVISWANATHAN, S. P., NISSIM, K., RASKHODNIKOVA, S., AND SMITH, A. Analyzing graphs with node differential privacy. In *TCC* (2013), pp. 457–476.
- [26] KATZ, J., AND LINDELL, Y. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2008.
- [27] KYROLA, A., AND GUESTRIN, C. Graphchi-db: Simple design for a scalable graph database system - on just a PC. *CoRR abs/1403.0701* (2014).
- [28] LESKOVEC, J., KLEINBERG, J. M., AND FALOUTSOS, C. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD* (2005), pp. 177–187.
- [29] LIU, K., AND TERZI, E. Towards identity anonymization on graphs. In *SIGMOD Conference* (2008), pp. 93–106.
- [30] LOW, Y., GONZALEZ, J., KYROLA, A., BICKSON, D., GUESTRIN, C., AND HELLERSTEIN, J. M. Graphlab: A new framework for parallel machine learning. In *UAI* (2010), pp. 340–349.
- [31] MALEWICZ, G., AUSTERN, M. H., BIK, A. J. C., DEHNERT, J. C., HORN, I., LEISER, N., AND CZAJKOWSKI, G. Pregel: a system for large-scale graph processing. In *SIGMOD* (2010), pp. 135–146.
- [32] MOURATIDIS, K., AND YIU, M. L. Shortest path computation with no information leakage. *PVLDB* (2012), 692–703.
- [33] NAVEED, M., PRABHAKARAN, M., AND GUNTER, C. Dynamic searchable encryption via blind storage. In *Oakland S&P 2014* (2014).

- [34] POTAMIAS, M., BONCHI, F., CASTILLO, C., AND GIONIS, A. Fast shortest path distance estimation in large networks. In *CIKM* (2009), pp. 867–876.
- [35] PRZULJ, N., WIGLE, D. A., AND JURISICA, I. Functional topology in a network of protein interactions. *Bioinformatics* 20, 3 (2004), 340–348.
- [36] QI, Z., XIAO, Y., SHAO, B., AND WANG, H. Toward a distance oracle for billion-node graphs. In *VLDB* (2013), pp. 61–72.
- [37] RIVEST, R., ADLEMAN, L., AND DERTOUZOS, M. On data banks and privacy homomorphisms. In *Foundations of Secure Computation* (1978), pp. 169–180.
- [38] SARWAT, M., ELNIKETY, S., HE, Y., AND KLIOT, G. Horton: Online query execution engine for large distributed graphs. In *ICDE* (2012), pp. 1289–1292.
- [39] SHANKS, D. Class number, a theory of factorization, and genera. In *1969 Number Theory Institute*. Providence, R.I., 1971, pp. 415–440.
- [40] SHAO, B., WANG, H., AND LI, Y. Trinity: a distributed graph engine on a memory cloud. In *SIGMOD* (2013), pp. 505–516.
- [41] SHEN, E., AND YU, T. Mining frequent graph patterns with differential privacy. In *KDD 2013* (2013), pp. 545–553.
- [42] SONG, D., WAGNER, D., AND PERRIG, A. Practical techniques for searching on encrypted data. In *Oakland S & P* (2000), pp. 44–55.
- [43] STEFANOV, E., PAPAMANTHOU, C., AND SHI, E. Practical dynamic searchable encryption with small leakage. In *Network and Distributed System Security Symposium (NDSS '14)* (2014).
- [44] THORUP, M., AND ZWICK, U. Approximate distance oracles. *Journal of the ACM* 52, 1 (Jan. 2005), 1–24.
- [45] VAN LIESDONK, P., SEDGHI, S., DOUMEN, J., HARTEL, P., AND JONKER, W. Computationally efficient searchable symmetric encryption. In *VLDB Conference on Secure Data Management* (Berlin, Heidelberg, 2010), SDM'10, pp. 87–100.
- [46] WANG, X. S., NAYAK, K., LIU, C., CHAN, T. H., SHI, E., STEFANOV, E., AND HUANG, Y. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014* (2014), pp. 215–226.