

# *cipherPath*: Efficient Traversals over Homomorphically Encrypted Paths

Georg Bramm<sup>a</sup> and Julian Schütte<sup>b</sup>

Fraunhofer AISEC, Lichtenbergstrasse 11, Garching near Munich, Germany

**Keywords:** Somewhat Homomorphic Encryption, Structured Encryption, Order Preserving Encryption, Geospatial Encryption, Dijkstra, Floyd, Shortest Path.

**Abstract:** We propose *cipherPath*, a novel graph encryption scheme that enables exact shortest distance queries on encrypted graphs. Shortest distance queries are very useful in a vast number of applications, including medical, social or geospatial. Our approach using somewhat homomorphic encryption in combination with structured encryption enables exact shortest distance queries on outsourced and encrypted graph data. Our approach upholds provable security against a semi-honest provider. We demonstrate our framework by means of two different shortest path algorithms on encrypted graphs: Dijkstra and Floyd. Finally, we evaluate the leakage profile of *cipherPath*.

## 1 INTRODUCTION


Nowadays service online map service providers tend to offer public and free services in order to be able to collect data from users trusting such a service. A very prominent example is Google Maps. While offering routing and shortest path services, they collect various information about your request. By collecting and aggregating those requests, an attacker might be able to identify your work or home place. Using *cipherPath*, we enable users to query such services without revealing the precise location. In order to achieve this, various new cryptographic techniques have been combined like structured encryption (SE), order preserving encryption (OPE) and somewhat homomorphic encryption (SHE). Utilizing these techniques, we can provide confidentiality for users in location-based services against an honest-but-curious provider. We want users being able to calculate shortest paths in real world applications without giving up their privacy concerns. We therefore present *cipherPath*, a new shortest path query processing algorithm over encrypted graphs in unstructured databases. Our algorithm guarantees the confidentiality of both the encrypted graph data and the query history against an honest-but-curious provider. To this end, we propose a structured encryption scheme utilizing somewhat homomorphic order preserving ciphertexts. Our contribution can be summarized as follows:


- We devise an interactive, yet very efficient SHE-OPE scheme, capable of adding, subtracting, multiplying, dividing, and comparing encrypted numbers.
- We present a generic framework of protocols for the generation, outsourcing and computation on an encrypted graph.
- We apply our framework to implement a classic shortest path finding technique by Dijkstra.
- We evaluate and discuss the information leakage of our framework.

After introducing some preliminaries we present our OPE-SHE scheme *ehOPE* in section 2. Building upon *ehOPE*, we introduce our framework *cipherPath* in section 3. After presenting a secure protocol for path finding in section 4, we discuss the leakage of our construction in section 5 and conclude in section 6.

## 2 FOUNDATION AND EXTENSIONS

We give a description of our notation and foundational preliminaries for this paper, followed by an introduction into *ehOPE*, an OPE-SHE scheme used in our framework.

<sup>a</sup>  <https://orcid.org/0000-0002-9020-5856>

<sup>b</sup>  <https://orcid.org/0000-0002-3007-6538>

## 2.1 Notation

$[n]$  donates the set of integers from zero up to  $n$ . A graph  $G = (V, E)$  consists of a set of vertices  $V$  and a set of edges  $E = \{(i, j)\}$  where  $i, j \in V$ . Any edge  $e = (i, j)$  with  $i, j \notin V$  is called  $\infty$  and is represented by randomly chosen integer values from a range that is invalid in the respective application domain<sup>1</sup>.

### 2.1.1 Distance Norm

As a distance metric we use the  $L_1^2$  norm on encrypted nodes, as it requires less computational and interaction overhead, compared to the  $L_2$  norm. For a real number  $p \geq 1$ , the p-norm or  $L_p$  distance norm of is defined by  $\|x\|_p = (|x_1|^p + |x_2|^p + \dots + |x_n|^p)^{1/p}$ . When we calculate the  $L_p$  distance  $d$  between the two nodes  $a$  and  $b$ , we write  $d = L_p(a, b)$ . When we calculate the  $L_1$  distance between a node  $b$  and the starting node, then we abbreviate this by writing  $d = dist(b)$ .

## 2.2 Cryptographic Preliminaries

We now introduce all necessary cryptographic preliminaries.

### 2.2.1 Pairing

Our work is partly based on the notion of *bilinear pairing maps*: suppose  $\mathbb{G}$  is an additive cyclic group and  $\mathbb{H}$  is a multiplicative cyclic group, both of the same order  $q$ . A bilinear pairing map  $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{H}$  satisfies the following properties:

- Bilinearity, i.e. for any  $P, Q \in \mathbb{G}$  and any  $a, b \in \mathbb{Z}_q^*$ , we have  $\hat{e}(P^a, Q^b) = \hat{e}(P^b, Q^a) = \hat{e}(P, Q)^{ab}$ .
- Non-degeneracy, which means that there exists a  $P, Q \in \mathbb{G}$ , such that  $\hat{e}(P, Q) \neq I \in \mathbb{H}$ , where  $I$  is the identity element.
- Computable, which means that for any  $P, Q \in \mathbb{G}$ , there exists an efficient algorithm to compute  $\hat{e}(P, Q) \in \mathbb{H}$ .

### 2.2.2 Induced Permutation

Let  $\pi$  be the permutation over  $[n]$  such that  $\forall i \in [n], m_i = Dec_K(c_{\pi(i)})$ , then we refer to  $\pi$  as the permutation *induced* by  $m$  and  $c$ . Accordingly, we refer to  $\Phi$  as the *permutation induced* by  $M$  and  $C$ , if  $\Phi$  is the permutation over  $[\lambda_1] \times [\lambda_2]$  such that

<sup>1</sup>For earth-bound navigation applications an integer in the interval  $]40075000, 2^{31} - 1]$  could be used, which is the earth's radius in meter and Integer max value.

<sup>2</sup>Sometimes also called *Manhattan* distance

$\forall (i, j) \in [\lambda_1] \times [\lambda_2], M_{(i,j)} = Dec_K(C_{\Phi(i,j)})$ . By inducing a pseudorandom permutation between  $m$  and  $c$  our construction is able to hide parts of the access pattern, as (Chase and Kamara, 2010) describe.

### 2.2.3 Somewhat Homomorphic Encryption

Besides hiding the structure of the graph we also need to hide the sequence of data values along a path from the server while doing calculations on it. We therefore integrated Pailliers scheme together with the ElGamal scheme into our own adoption of hOPE originally written by (Peng et al., 2017).

**Paillier.** The Paillier scheme (Paillier, 1999) allows additive homomorphic operations on ciphertexts in a probabilistic asymmetric encryption scheme. The scheme can be modified to allow signed values and subtraction by reducing the range of  $m$  to  $-\frac{n}{4} < m < \frac{n}{4}$ . The modified Paillier cryptosystem has the following properties:

- Addition: The product of two ciphertexts  $ct_1 = E_k(m_1)$  and  $ct_2 = E_k(m_2)$  results in the encryption of the sum of their plaintexts:  $(ct_1 \times ct_2) \bmod n^2 = E_k(m_1 + m_2)$
- Subtraction: The product of a ciphertext  $ct_1 = E_k(m_1)$  with the modular inverse of another ciphertext  $ct_2 = E_k(m_2)$  computed modulo  $n^2$  results in the encryption of the subtraction of their plaintexts:  $(ct_1 \times ct_2^{-1}) \bmod n^2 = E_k(m_1 - m_2)$
- Constant multiplication: The  $b^h$  power of ciphertext  $ct_1 = E_k(m_1)$  results in the encryption of the product of  $b$  and  $m_1$ :  $E_k(m_1 \times b) = E_k(m_1)^b \bmod n^2$
- Semantic security: The generated ciphertexts are non-deterministic.

**ElGamal.** In 1985 Taher ElGamal published a scheme with multiplicative homomorphic properties (ElGamal, 1985). Besides multiplication, ElGamal also allows division by multiplying with the modular inverse. As (Lipmaa, 2010) state, the ElGamal cryptosystem is *IND - CCA1* and has the following properties:

- Multiplication: The product of two ciphertexts  $ct_1 = E_k(m_1)$  and  $ct_2 = E_k(m_2)$  results in the encryption of the product of their plaintexts:  $(ct_1 \times ct_2) = E_k(m_1 \cdot m_2)$ .
- Division: The product of a ciphertext  $ct_1 = E_k(m_1)$  with the multiplicative inverse of another ciphertext  $ct_2 = E_k(m_2)$  results in the encryption of the division of their plaintexts:  $\frac{ct_1}{ct_2} = (ct_1 \cdot (ct_2)^{-1}) = E_k(\frac{m_1}{m_2})$

- Semantic security: The generated ciphertexts are non-deterministic.

### 2.3 homomorphic Order Preserving Encryption (hOPE)

A new technique called hOPE was introduced by (Peng et al., 2017). The authors combine Pailliers scheme with an OPE approach based on a  $B^+$  code tree. In cipherPath we need to compute additions, subtractions, multiplications, as well as inequality tests, so we extend hOPE by also incorporating El-Gamal. The additional interactivity with the client in our scheme is needed when translating values from one SHE to the other, like in  $Add(SP, L_1, L_2)$  (5). The encrypted value is given to the client, who decrypts it using  $K$ , encrypts it into the opposite SHE scheme and uploads it again to the server. We achieve the same mathematical expressiveness as the Peng's *PhOPE*, but without integrating a fully homomorphic encryption (FHE) scheme. Instead we integrate two SHE schemes. Our construction *ehOPE* is given by:

Select a bilinear pairing  $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{H}$  with the order  $q \in \mathbb{Z}^*$  for  $\mathbb{G}$  and  $\mathbb{H}$ . Pick an additive  $\Pi^+ = (\text{Gen}, \text{Enc}, \text{Dec}, \text{Add}, \text{Sub})$  and a multiplicative SHE scheme  $\Pi^* = (\text{Gen}, \text{Enc}, \text{Dec}, \text{Mul})$ . Then  $ehOPE = (\text{Setup}, \text{Gen}, \text{Enc}, \text{Dec}, \text{Add}, \text{Sub}, \text{MulDiv}, \text{Comp})$  is defined by:

**Gen()**: Generate key pairs for both SHE schemes  $\{EK^+, DK^+\} \leftarrow \Pi_{Gen}^+(1^k)$  and  $\{EK^*, DK^*\} \leftarrow \Pi_{Gen}^*(1^k)$  and return them as key pair  $\{SK, PK\}$ , with  $SK = \{DK^+, DK^*\}$  and  $PK = \{EK^+, EK^*\}$ .

**Enc(SP, EK, m)**: Ciphertext  $CT = \langle p, e, G, H, o \rangle$  is obtained as follows:

- (1) Using  $EK$  compute  $p = \Pi_{Enc}^+(m)$ ,  $e = \Pi_{Enc}^*(m)$ ,  $G = m \cdot P \in \mathbb{G}$  and  $H = \hat{e}(P, P)^m \in \mathbb{H}$ .
- (2) Look up  $L_i = \langle p_i, e_i, G_i, H_i, o_i \rangle$  in  $\mathbb{L}$  by matching  $G$  or  $H$ . If found, skip to (3); otherwise skip to (4).
- (3) Return  $CT = L_i = \langle p_i, e_i, G_i, H_i, o_i \rangle$ , where  $G_i = G$  or  $H_i = H$ . The process is now terminated.
- (4)  $o$  is determined using an interactive code location algorithm based on tree  $\mathcal{T}$ .
- (5) Insert  $L = \langle p, e, G, H, o \rangle$  into the cache  $\mathbb{L}$ , and insert  $c$  into  $\mathcal{T}$ . Then update both and go to (2).

**Add(SP,  $L_1, L_2$ )**: Take as input  $SP$ ,  $L_1 = \langle p_1, e_1, G_1, H_1, o_1 \rangle$  and  $L_2 = \langle p_2, e_2, G_2, H_2, o_2 \rangle$ . The sum  $L = L_1 + L_2$  is generated as follows:

- (1) Compute  $p = p_1 + p_2$ ,  $G = G_1 + G_2$  and  $H = \hat{e}(G, P)$ .
- (2) Look up  $L_i = \langle p_i, e_i, G_i, H_i, o_i \rangle$  in  $\mathbb{L}$  by matching  $G$  or  $p$ . If found, skip to (3); otherwise, skip to (4).
- (3) ,(4) Same as in Enc (3) and Enc (4).
- (5) ask client to translate  $p$  to  $e$ , i.e. calculate  $e = \Pi_{Enc}^+(\Pi_{Dec}^+(p))$ .
- (6) Same as in Enc (5).

**Sub(SP,  $L_1, L_2$ )**: Take as input  $SP$ ,  $L_1 = \langle p_1, e_1, G_1, H_1, o_1 \rangle$  and  $L_2 = \langle p_2, e_2, G_2, H_2, o_2 \rangle$ . The subtraction  $L = L_1 - L_2$  is generated as follows:

- (1) Compute  $p = p_1 - p_2$ ,  $G = G_1 + (G_2)^{-1}$  and  $H = \hat{e}(G, P)$ .

- (2) Look up  $L_i = \langle p_i, e_i, G_i, H_i, o_i \rangle$  in  $\mathbb{L}$  by matching  $G$  or  $H$ . If found, skip to (3); otherwise, skip to (4).
- (3) ,(4) Same as in Enc (3) and Enc (4).
- (5) ask client to translate  $p$  to  $e$ , i.e. calculate  $e = \Pi_{Enc}^+(\Pi_{Dec}^+(p))$ .
- (6) Same as in Enc (5).

**Mul(SP,  $L_1, L_2$ )**: Take as input  $SP$ ,  $L_1 = \langle p_1, e_1, G_1, H_1, o_1 \rangle$  and  $L_2 = \langle p_2, e_2, G_2, H_2, o_2 \rangle$ . The multiplication  $L = L_1 \cdot L_2$  is generated as follows:

- (1) Compute  $e = e_1 \cdot e_2$  and  $H = \hat{e}(G_1, G_2)$ .
- (2) Look up  $L_i = \langle p_i, e_i, G_i, H_i, o_i \rangle$  in  $\mathbb{L}$  by matching  $H$ . If found, skip to (3); otherwise, skip to (4).
- (3) ,(4) Same as in Enc (3) and Enc (4).
- (5) Ask client to compute  $G = \Pi_{Dec}^*(e) \cdot P$  and translate  $p = \Pi_{Enc}^+(\Pi_{Dec}^*(e))$ .
- (6) Same as in Enc (5).

**Div(SP,  $L_1, L_2$ )**: Take as input  $SP$ ,  $L_1 = \langle p_1, e_1, G_1, H_1, o_1 \rangle$  and  $L_2 = \langle p_2, e_2, G_2, H_2, o_2 \rangle$ . The division  $L = \frac{L_1}{L_2}$  is generated as follows:

- (1) Compute  $e = e_1 \cdot e_2^{-1}$  and  $H = \hat{e}(G_1, G_2)^{-1}$ .
- (2) Look up  $L_i = \langle p_i, e_i, G_i, H_i, o_i \rangle$  in  $\mathbb{L}$  by matching  $H$ . If found, skip to (3); otherwise, skip to (4).
- (3) ,(4) Same as in Enc (3) and Enc (4).
- (5) Ask client to compute  $G = \Pi_{Dec}^*(e) \cdot P$  and translate  $p = \Pi_{Enc}^+(\Pi_{Dec}^*(e))$ .
- (6) Same as in Enc (5).

**COMP( $L_1, L_2$ )**: Take as input  $L_1$  and  $L_2$ . Output the result of the binary comparison  $o_1$  and  $o_2$ . Return 0 if they are equal. Returns 1 if  $L_1 \geq L_2$  or  $-1$  if  $L_1 \leq L_2$ .

**Dec(DK,  $L_1$ )**: Take as input  $L_1 = \langle p_1, e_1, G_1, H_1, o_1 \rangle$ . Output  $m = \Pi_{Dec}^+(p_1)$ .

The security analysis for hOPE is given by (Peng et al., 2017). As long as we pick at least  $IND - O2CPA$  secure SHE schemes, our modified scheme upholds the same security features as the FHE version, with the disadvantage of more interactivity and the advantage of a higher efficiency.

### 2.4 Related Work

Searchable encryption was initially introduced by (Song et al., 2000). They propose a novel, provably secure scheme that enables searching on encrypted data without decrypting it. Their technique is based on a stream cipher. Many secure searchable encryption schemes followed like (Curtmola et al., 2011), (Cash et al., 2014) or (Bost, 2016). The schemes were mainly focusing on text based queries. (Meng et al., 2015) proposed GRECS, a encrypted graph scheme that supported very efficient, but only approximate, shortest distance queries. It was a big step forward, but parts of the calculation, the intermediate distance matching, is still calculated on the client side. Distances were precalculated and only approximate. Other approaches to calculate paths on encrypted graphs include attempts to use secure multiparty computation (SMC), like (Failla, 2010). In this scenario two parties try to find the shortest path in a

privacy preserving way. One party knows the weights on the edges of the graph, the other party knows an heuristic to find the best path and together they try to solve the quest. As each party must have information about the graph or the heuristic, before any query is sent, this approach is not comparable to ours. In 2015, (Samanthula et al., 2015) also tried to approach this problem by applying SHE to adjacency lists, instead of an adjacency matrix as in our case. Another huge difference in their approach is the calculation of path subsets on the client side, instead of the server side. (Chase and Kamara, 2010) showed a novel way to construct a scheme that was taking care not only of the data values, but the structure of the adjacency matrix as well. The structured encryption scheme showed a way how to efficiently and privately query encrypted graphs, while at the same time keeping the data and the structure hidden. Structured encryption, which generalizes previous work on symmetric searchable encryption (SSE) to the setting of arbitrarily-structured data is, besides *ehOPE*, a foundational building block of this paper.

### 3 FRAMEWORK

Our framework *cipherPath* is based on multiple modifications of the associative structured encryption scheme for labeled data by Chase. It was modified in order to incorporate SHE operations on ciphertexts instead of simple symmetrically encrypted ciphertexts. Besides changes in the token and lookup algorithms, the replacement of symmetrically encrypted ciphertexts with *ehOPE* ciphertexts is a fundamental change. This allows us to keep path lengths confidential, while still being able to compute on them.

#### 3.1 Scheme Setting

The intention of our work is to keep the graph, the shortest path queries as well as the results hidden from an attacker. If some client itself holds the entire graph, he could run various path searching algorithms locally, way faster. But this implies that he holds all the necessary resources, including the graph data as well as the computational power. In a setting with multiple mobile lightweight clients, like smartphones or tablets, we advise the following procedure: A honest but curious provider sets up the scheme and generates *SP* using *cipherPath.Setup*. Afterwards the provider makes *SP* publicly available. A trusted entity generates a single key *K* using *cipherPath.Gen* and encrypts the routing graph *G* regularly using *cipherPath.Enc*. The trusted entity uploads the generated *CT* regularly

to the provider. The trusted entity gives out key *K* to multiple, eligible users over a secure channel. The eligible users generate queries  $Q_i$  using *K* and *SP* and query the scheme using *DiJ(Q<sub>i</sub>)*. Finally the eligible users can decrypt the routing result locally using *K*. In such a setting the query leakage is kept low by regularly renewing *CT* and thus rendering older leakage useless. The users are able to query the system efficiently and their privacy concerns is taken care of.

#### 3.2 Information Model

We need a way to not only store our permuted and encrypted edges, but also information on the permuted and encrypted nodes. Therefore we use the encrypted edge matrix *C* and an encrypted node vector *D*. The message space of our main framework takes as input a matrix *M* and a set of nodes *N* together with two sequences *m* and *n* consisting of  $((x_1, l_1), \dots, (x_n, l_n))$ , with a data item  $x_i$  and a label  $l_i$ . In case of *m*, a data item is composed of an identifier and a length. In case of *n*, a data item is composed of an identifier and a coordinate composed of a longitude and latitude value.

#### 3.3 Scheme Architecture

In order to enable queries for both edges and nodes, we adopted the corresponding algorithms that are either index based ( $T_I$ ) or label based ( $T_L$ ). Likewise, lookup algorithms are also available for both types of token requests. In a real world location-based scenario, we are dealing with two different kinds of arithmetic data in a navigation graph, namely node coordinates, composed of longitude and latitude values, given as double precision floating point values on the one hand and edge lengths in meters or minutes as integer values on the other hand. As there is no reason to compare lengths and coordinates, we decided to include in our framework two *ehOPE* schemes for each data type: *ehOPE<sup>R</sup>* for floating point values and *ehOPE<sup>Z</sup>* for integer values.

##### 3.3.1 Setup

This algorithm runs once on the provider side and generates the system parameters (*SP*) for the *ehOPE* schemes depending on security parameter *k*. Setup is run by the provider, as he has to set up cache  $\mathbb{L}$  and code tree *T*. The random value of *P* might also be chosen by the client, but this is irrelevant, as *SP* is public knowledge.



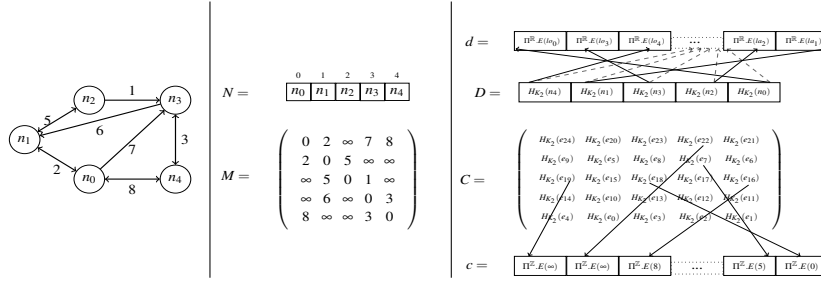


Figure 1: A graphical representation of steps (1), (3) and (4) during the encryption of a graph in *cipherPath*.

### 3.3.2 Key Generation

While  $SP$  is public knowledge,  $K$  must be kept secret.  $K$  is composed of  $K_1$ , the controlled disclosure key,  $K_2$  the permutation key, and  $K_3, K_4$  the object keys. The key  $K$  is used to encrypt a graph at the client side, as well as to decrypt single object values, i.e. lengths or coordinates. As mentioned in section 2.2.3, the range of  $m$  is reduced to  $-\frac{k}{4} < m < \frac{k}{4}$ .

### 3.3.3 Encryption

Encryption in *cipherPath* means to encrypt and permute edges and nodes of a graph in such a way that only  $L_{Enc}$  and  $L_{Query}$  will be leaked to the server. Further information about the graph, the query shell be hidden from the provider completely. The length of an edge is encrypted using an *ehOPE* scheme responsible for integer values called  $\Pi^{\mathbb{Z}}$  and is linked to a permuted entry in  $C$ , as in the algorithm *cipherPath.Enc*( $SP, K, M, N$ ) step (3). The longitude ( $lo$ ) and latitude ( $la$ ) values are encrypted using  $\Pi^{\mathbb{R}.Enc}$  a second floating point instance of *ehOPE*. Nodes are then linked to two permuted entries in  $D$  for the  $lo$  and  $la$  value, as can be seen in step (4). Note, that permutation  $Q$  of vector  $D$  and permutation  $P$  of matrix  $D$  result in the same arrangement, when comparing the order of the vector to the order of the rows. Finally all ciphertext elements, matrix  $C$  and its values  $c_i$  as well as vector  $D$  and its values  $d_i$  are returned as ciphertext ( $CT$ ).

### 3.3.4 Token

In a real world scenario no one will know the exact index of a specific edge or node of a graph, whilst street (edge) or building (node) names are easy to remember. Therefore we introduce a trapdoor  $T_L$  for label based searches. By embedding an object identifier together with a hashed and row- and column-wise encrypted label inside the matrix  $C$  we are able to index all edges. The node vector also embeds two identifiers ( $lo, la$ ) together with a hashed and index-wise encrypted label in  $D$ .

### 3.3.5 Lookup

In the lookup algorithms we parse the index ( $T_I$ ) or label based ( $T_L$ ) token calculate the pointer  $i$  to the correct position in  $c$  respectively  $d$ . Afterwards we return the pointer together with the object ciphertext ( $\{CT_i, i\}$ ). This behavior can be seen in Figure 2. The matrix  $C$  and the vector  $D$  can be thought of a kind of a searchable encrypted index with pointers to the corresponding data items  $c_i$  and  $d_i$ .

### 3.3.6 Decryption

To decrypt  $c_i$  or  $d_i$  the client utilizes the corresponding decrypt functions  $Dec^{\mathbb{Z}}(SP, K, c_i)$  for integer values or  $Dec^{\mathbb{R}}(SP, K, d_i)$  for floating point values.

## 4 SHORTEST PATH

The shortest path problem is sometimes also called single-source shortest path problem (SSSP), to distinguish it from other variations like the single-destination shortest path (SDSP) or the all-pairs shortest path problem (APSP). Before we inspect (Dijkstra et al., 1959) on *cipherPath*, we want to make clear that it is also possible to take the APSP approach, for example using (Floyd, 1962) on *cipherPath*. Additional sets  $d$  for distances,  $p$  for predecessors as well as  $Q$  are needed to run the algorithm by Dijkstra, as can be seen in Figure 3. First of all the label based trapdoors for start (1) and end (2) need to be looked up. After an initialization of  $d$  (1,3),  $p$  (3,4) and  $Q$  (4) we look for the shortest path of the current node to another node  $u$  (4.a) until we reached our target (4.b) or there are no more nodes left (4).

Otherwise we inspect each neighbor of  $u$  by asking the client to disclose the  $u^{th}$  row in  $CT.C$  (5.c.I) and looking up the corresponding lengths (5.c.II). Using the disclosed information we are able to run *update* (5.c.III). As we are proceeding to disclose nodes on the path between  $T_L^S$  and  $T_L^T$ , we are leaking information that is helping the attacker.

Let  $H : \{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^h$  and  $F : \{0, 1\}^k \times \{0, 1\}^* \rightarrow \{0, 1\}^k$  be pseudo-random functions,  $Q : \{0, 1\}^k \times [n] \rightarrow [n]$  and  $P : \{0, 1\}^k \times [\lambda_1] \times [\lambda_2] \rightarrow Q(K, [\lambda_1]) \times Q(K^{-1}, [\lambda_2])$  be pseudo-random permutations and let  $\Pi^{\mathbb{Z}} = (\text{Setup}, \text{Gen}, \text{Enc}, \text{Dec}, \text{Add}, \text{Sub}, \text{Mul}, \text{Comp})$  be an  $ehOPE^{\mathbb{Z}}$  and  $\Pi^{\mathbb{R}} = (\text{Setup}, \text{Gen}, \text{Enc}, \text{Dec}, \text{Add}, \text{Sub}, \text{Mul}, \text{Comp})$  be an  $ehOPE^{\mathbb{R}}$  encryption scheme. Our encrypted shortest path framework  $cipherPath = (\text{Setup}, \text{Gen}, \text{Enc}, \text{Token}_I, \text{Token}_L, \text{Lookup}_I, \text{Lookup}_L, \text{Dec})$  is then defined as follows:

$\{SP\} \leftarrow \mathbf{Setup}(1^k, h)$ : Setup both ehOPE schemes  $\{SP\} \leftarrow \{\Pi^{\mathbb{R}}.\text{Setup}(1^k), \Pi^{\mathbb{Z}}.\text{Setup}(1^k), h, h_1, h_2\}$  with  $k$  being the security parameter and  $h = h_1 + h_2$  (with  $h_1$  and  $h_2$  being the bit length of pointers to  $d$  and  $c$ ) Return the SP.

$\{K\} \leftarrow \mathbf{Gen}(SP)$ : generate two random  $k$ -bit strings  $K_1, K_2$  and key  $K_3 = (EK, DK) \leftarrow \Pi^{\mathbb{R}}.\text{Gen}()$  and  $K_4 = (EK, DK) \leftarrow \Pi^{\mathbb{Z}}.\text{Gen}()$ . Output  $K = (K_1, K_2, K_3, K_4)$ .

$\{CT\} \leftarrow \mathbf{Enc}(SP, K, M, N)$ : Construct ciphertext  $CT$  as follows:

(1) Parse the edge matrix  $\mathbf{M}$  as  $m$  with label  $l_m$  and parse the node vector  $N$  as  $n$  with label  $l_n$ .

(2) Choose a  $k$ -bit random key  $K_4$ .

(3) Generate matrix  $C$  and vector  $c$  by:

(a) Choose a pseudo-random permutation  $G : \{0, 1\}^k \times [m] \rightarrow [m]$ .

(b) Generate a  $\lambda_1 \times \lambda_2$  matrix  $C$  as follows:

$\forall (\alpha, \beta) \in [\lambda_1] \times [\lambda_2]$ : store  $\langle G_{K_4}(i), H_{K_2}(l_i) \rangle \oplus F_{K_1}(\alpha, \beta)$  where  $i = M_{\alpha, \beta}$  at  $(\alpha', \beta') = P_{K_2}(\alpha, \beta)$  in  $C$ .

(c) If  $M_{\alpha, \beta} = \perp$ , then  $\langle G_{K_4}(i), H_{K_2}(l_i) \rangle$  is replaced with a random, large enough value  $\langle \infty, H_{K_2}(\infty) \rangle$ .

(d)  $\forall (i) \in [m]$ : Let  $j = G_{K_4}(i)$  and set  $c_i \leftarrow \Pi^{\mathbb{Z}}.\text{Enc}_{K_3}(m_j)$ .

(4) Generate index vector  $D$  of length  $n$  and data vector  $d$  of length  $2n$ :

(a) Choose a pseudo-random permutation  $I : \{0, 1\}^k \times [n] \rightarrow [n]$ .

(b) Generate vector  $D$  of length  $n$  as follows:

$\forall (i) \in [n]$ : store  $\langle I_{K_4}(i), I_{K_4}(i+n), H_{K_2}(l_i) \rangle \oplus F_{K_1}(i)$  at location  $n' = Q_{K_2}(i)$  in  $D$

(c) Generate vector  $d$  of length  $2n$  as follows:

$\forall (i) \in [n]$ : Let  $j = I_{K_4}(i)$  and set  $d_i \leftarrow \Pi^{\mathbb{R}}.\text{Enc}_{K_3}(n_i^{lo})$ .

$\forall (i) \in [n]$ : Let  $j = I_{K_4}(i+n)$  and set  $d_j \leftarrow \Pi^{\mathbb{R}}.\text{Enc}_{K_3}(n_i^{la})$ .

(5) Output  $CT = \{C, c, D, d\}$ .

$\{T_L\} \leftarrow \mathbf{Token}_L(SP, K, l)$ : generate  $\forall (i) \in [n]$ : set  $t_i = \langle 0, 0, H_{K_2}(l) \rangle \oplus F_{K_1}(i)$ . Output  $T_L = \{t_0, t_1, \dots, t_n\}$ .

$\{T_L\} \leftarrow \mathbf{Token}_L(SP, K, l)$ : generate  $\forall (\alpha, \beta) \in [\lambda_1] \times [\lambda_2]$ : set  $t_i = \langle 0, 0, H_{K_2}(l) \rangle \oplus F_{K_1}(\alpha, \beta)$ . Output  $T_L = \{t_0, t_1, \dots, t_m\}$ .

$\{T_I\} \leftarrow \mathbf{Token}_I(SP, K, \alpha, \beta)$ : output  $T_I = (s, \alpha', \beta')$ , where  $s = F_{K_1}(\alpha, \beta)$  and  $(\alpha', \beta') = P_{K_2}(\alpha, \beta)$ .

$\{T_I\} \leftarrow \mathbf{Token}_I(SP, K, \alpha)$ : output  $T_I = (s, \alpha', \perp)$ , where  $s = F_{K_1}(\alpha)$  and  $\alpha' = Q_{K_2}(\alpha)$ .

$\{CT_*, i\} \leftarrow \mathbf{Lookup}_I(SP, CT, T_I)$ : parse  $T_I$  as  $(s, \alpha', \beta')$ ;

(1) If  $\beta' = \perp$ , this is a node token:

compute  $i = (s \oplus CT.D_{\alpha'}) \gg (h_2 + h_1)$  and  $j = (s \oplus CT.D_{\alpha'}) \gg h_1$  and output  $\{\{CT.d_i, i\}, \{CT.d_j, j\}\}$ .

(2) If there is a  $\beta'$  value, compute  $j = (s \oplus CT.C_{\alpha', \beta'}) \gg h$  and output  $\{CT.c_j, j\}$ .

$\{CT_*, i\} \leftarrow \mathbf{Lookup}_L(SP, CT, T_L)$ : parse  $T_L$  as  $(t_0, t_1, \dots, t_x)$ ;

(1) If  $x = n$ , this is a node lookup. Compute  $\forall (i) \in [n]$ :

(a) compute  $h = CT.D_i \ll (k - h)$ .

(b) if  $h = (t_i \ll (k - h))$  compute  $j = (t_i \oplus CT.D_i) \gg (h_2 + h_1)$  and  $k = (t_i \oplus CT.D_i) \gg h_1$  and output  $\{i, \{CT.c_j, j\}, \{CT.c_k, k\}\}$ .

(c) Otherwise there is no node. output  $\{\perp, \perp\}$ .

(2) Otherwise this is an edge lookup. Compute  $\forall (i) \in [m]$ :

(a) compute  $\alpha' = (i \bmod n)$  and  $\beta' = \lfloor \frac{i}{n} \rfloor$

(b) compute  $h = CT.C_{\alpha', \beta'} \ll (k - h)$ .

(c) if  $h = t_i \ll (k - h)$ : compute  $j = (t_i \oplus CT.C_{\alpha', \beta'}) \gg h$  and output  $\{CT.c_j, j\}$ .

$\{PT_V^{lo}, PT_V^{la}\} \leftarrow \mathbf{Dec}^{\mathbb{R}}(SP, K, CT_V)$ : Decrypt vertex coordinates  $PT_V^{lo} = \Pi^{\mathbb{R}}.\text{Dec}_{K_3}(CT_V^{lo})$  and  $PT_V^{la} = \Pi^{\mathbb{R}}.\text{Dec}_{K_3}(CT_V^{la})$ .

$\{PT_E\} \leftarrow \mathbf{Dec}^{\mathbb{Z}}(SP, K, CT_E)$ : Decrypt edge length  $CT_E$  using  $\Pi^{\mathbb{Z}}.\text{Dec}_{K_4}(CT_E)$ .

Figure 2: Steps Setup, Gen, Enc, Token, Lookup, Dec of our encrypted geospatial framework.

$\{(CT_1^Z, n_1), \dots, (CT_T^Z, n_T)\} \leftarrow \text{Dij}(SP, CT, T_L^S, T_L^T)$ :  
 (1)  $\{CT^T, s\} = \text{cipherPath.Lookup}_I(SP, CT, T_L^S)$ ,  
 $d[s] = 0$ .  
 (2)  $\{CT^S, t\} = \text{cipherPath.Lookup}_I(SP, CT, T_L^T)$ .  
 (3)  $\forall n \in CT.D \setminus \{s, t\}$ :  
 $d[n] = \infty$ ,  $p[n] = \perp$ . add  $n$  to  $Q$ .  
 (4) While  $Q \neq \emptyset$ :  
 (a)  $u \leftarrow \min_{q \in Q}(Q)$ .  
 (b) If  $u == t$  return  $p[]$ .  
 (c) Otherwise remove  $u$  from  $Q$  and  
 (I) disclose  $T = \{T_I^1, \dots, T_I^n\} \leftarrow \forall i \in [n] :$   
 $\text{cipherPath.Token}_I(SP, K, u, i)$ .  
 (II)  $CT = \{\{CT^0, i_0\}, \dots, \{CT^n, i_n\}\} \leftarrow \forall i \in T :$   
 $\text{cipherPath.Lookup}_I(SP, CT, T_I^i)$ .  
 (III)  $\forall i_0, \dots, i_n \in CT : \text{If } n_i \in Q :$   
 $a \leftarrow \Pi^Z.Add(SP, d[u], \text{dist}(u, v))$   
 If  $\Pi^Z.Comp(a, d[v]) == -1$ : Set  $d[v] := a$  and  
 $p[v] := u$ .

Figure 3: Dijkstra based on *cipherPath*.

## 5 LEAKAGE

We now present our simulation-based security definition against malicious adversaries. Please recall the definition of  $(\mathcal{L}_{Enc}, \mathcal{L}_{Query}) - \text{CQA2}$  security. In a CQA2 attack, we assume that the adversary is allowed to make multiple adaptive queries to the data structure. We seek to guarantee that the adversary can only learn what was accessed and the result of the query. All other information is to be kept secret.

### Theorem 5.1.

Let  $\Lambda = (\text{Setup}, \text{Gen}, \text{Enc}, \text{Token}_*, \text{Lookup}_*, \text{Dec}^*)$  be an *cipherPath* scheme for message space  $m$ , query space  $q$ , result space  $r$  and the query  $F : m \times q \rightarrow r$ . Consider the following two experiments for the leakage functions  $\mathcal{L}_{Enc}$  and  $\mathcal{L}_{Query}$ , adversary  $A$ , challenger  $C$ , and simulator  $S$ :

$\text{Real}_{A,C}^\Lambda(\lambda)$ :  $C$  begins by running  $\Lambda.Gen(1^\lambda)$  to generate  $K$ .  $A$  outputs a graph  $\mathcal{G}$ .  $C$  runs  $\Lambda.Enc(K, \mathcal{G})$  to generate  $CT$ . Afterwards he sends  $CT$  to  $A$ .  $A$  adaptively makes a polynomial number of Dijkstra queries  $q_0, \dots, q_t$ . For each query  $q_i$ ,  $A$  interacts with  $C$ .  $C$  plays the part of the client in the protocol with input  $(K, q_i)$  and sends its output to  $A$ . Finally,  $A$  outputs a bit  $b$ .

$\text{Ideal}_{A,C,S}^\Lambda(\lambda)$ : Initially,  $A$  outputs graph  $\mathcal{G}$ .  $S$  is given  $\mathcal{L}_{Enc}(\mathcal{G})$ , and outputs  $CT$ .  $A$  adaptively makes a polynomial number of queries  $q_0, \dots, q_t$ . For each query  $q_i$ ,  $S$  is given  $\mathcal{L}_{Query}(m, q_0, \dots, q_i)$  and interacts with  $A$ .  $S$  produces a flag  $f$ ; if  $f = \emptyset$ , the challenger sends  $\emptyset$  to  $A$ , otherwise it sends  $F(m, q_i)$ . Finally,  $A$  outputs a bit  $b$ .

We say that  $\Lambda$  is  $(\mathcal{L}_{Enc}, \mathcal{L}_{Query}) - \text{CQA2}$  secure against

malicious adversaries if, for all PPT adversaries  $A$ , there exists a simulator  $S$  such that:

$$|\Pr[\text{Real}_{A,C}^\Lambda(\lambda) = 1] - \Pr[\text{Ideal}_{A,C,S}^\Lambda(\lambda) = 1]| \leq \text{negl}(\lambda)$$

The information that is leaked about the message  $m$  is given by  $\mathcal{L}_{Enc}(m)$  and is analyzed in subsection 5.1. The shortest path query also leaks information, which is given by  $\mathcal{L}_{Query}(m, q_0, \dots, q_n)$ . It denotes the leakage of  $\mathcal{L}_{Enc}(m)$  in combination with all executed queries from  $(q_0)$  up to now  $(q_n)$ . This behavior is analyzed in subsection 5.2.

### 5.1 Leakage Analysis of Encryption

The encryption leakage  $\mathcal{L}_{Enc}$  of our construction outputs  $(N, \mathcal{O}_{1D}(CT.d), \mathcal{O}_{1D}(CT.c))$ , which is defined by:

- the total number of encrypted nodes  $N = |d|$  in the underlying graph  $\mathcal{G}$ .
- the 1D order of all edges by length, given as  $\mathcal{O}_{1D}(CT.c) = \text{Ord}_{1D}(c_x < \dots < c_y)$ .
- the 1D order of all coordinate values  $lo$  and  $la$ , given as  $\mathcal{O}_{1D}(CT.d) = \text{Ord}_{1D}(d_x < \dots < d_y)$ .

The order of all values of  $\Pi^Z$  and  $\Pi^R$  is leaked on the server, but as long as the server is not able to correlate those values to nodes or edges we are safe.

### 5.2 Leakage Analysis for Queries

The probability to reconstruct  $\mathcal{G}$  is higher, when additional leakage by the query algorithms can be correlated to leaked information by  $\mathcal{L}_{Enc}(\mathcal{G})$ . As Dijkstra and Floyd both intentionally leak the 2D as well as the path pattern, including the  $L_1$  distance between multiple nodes along the path, an honest but curious adversary could collect this information to further refine his knowledge about the plaintext graph  $\mathcal{G}$ . The query leakage  $\mathcal{L}_{Query}$  of our construction gives  $(s, \mathcal{O}_{2D}(n), e)$ , which is defined by:

- the length pattern  $s$ , which is the number of edges and nodes between the start and the target node.
- the 2D order of all nodes  $n$  along a path  $\mathcal{P}$ , given as by  $\mathcal{O}_{2D}(n) = \text{Ord}_{2D}(\{n_S, \dots, n_T\}) = \text{Ord}_{1D}(n_S^{lo} < \dots < n_T^{lo}) \times \text{Ord}_{1D}(n_S^{la} < \dots < n_T^{la})$ . This is leaked, because the server is now able to correlate the 2D order of nodes along  $\mathcal{P}$  by their  $lo$  and  $la$  order.
- the edge pattern  $e$  along a path  $\mathcal{P}$ , which allows an attacker to link the 1D order of edges to specific nodes. This information helps the attacker to even further refine his information about  $\mathcal{G}$ .

As there is no frequency in  $\Pi^+$  and  $\Pi^*$ , we leak the ideal profile according to (Durak et al., 2016). The order of the coordinates may be abused to reconstruct  $\mathcal{G}$  in a 2-D sort attack. By the metrics of (Naveed

et al., 2015), such an attack does not work a hundred percent, yet it is obvious that much of the structural behavior is revealed, including relatively fine details like the density of points in specific areas of the graph. In order to prevent such an attack, we can restrict the number of nodes. As *CT* is only used for routing and not display purposes, we can reduce the attack vector by limiting the number of encrypted nodes, while parsing an OSM map. This can be done by throwing away unnecessary intermediate nodes, for example nodes with a degree  $\leq 2$ .

## 6 CONCLUSION

We presented *cipherPath*: Efficient Traversals over Homomorphically Encrypted Paths, a framework for the computation of the shortest path in an encrypted graph. We showed how to construct our framework based on cryptographic preliminaries and how find the shortest paths between encrypted nodes. Finally we analyzed the security and the leakage of our construction. A future direction might be the defense against more sophisticated graph similarity attacks, like the neural network approach given by (Bai et al., 2018). Our goal is to find an upper barrier on the number of nodes, from which on neural network attacks become feasible. Another direction of research might be the sharding of a *CT* into multiple subsets spread across multiple provider.

## ACKNOWLEDGEMENTS

This work has been funded by the Fraunhofer Cluster of Excellence 'Cognitive Internet Technologies'<sup>3</sup>.

## REFERENCES

- Bai, Y., Ding, H., Sun, Y., and Wang, W. (2018). Convolutional set matching for graph similarity. *arXiv preprint arXiv:1810.10866*.
- Bost, R. (2016).  $\Sigma$  οφός: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1143–1154.
- Cash, D., Jaeger, J., Jarecki, S., Jutla, C. S., Krawczyk, H., Rosu, M.-C., and Steiner, M. (2014). Dynamic searchable encryption in very-large databases: data structures and implementation. In *NDSS*, volume 14, pages 23–26. Citeseer.
- Chase, M. and Kamara, S. (2010). Structured encryption and controlled disclosure. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 577–594. Springer.
- Curmola, R., Garay, J., Kamara, S., and Ostrovsky, R. (2011). Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934.
- Dijkstra, E. W. et al. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.
- Durak, F. B., DuBuisson, T. M., and Cash, D. (2016). What else is revealed by order-revealing encryption? In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1155–1166.
- ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472.
- Failla, P. (2010). Heuristic search in encrypted graphs. In *2010 Fourth International Conference on Emerging Security Information, Systems and Technologies*, pages 82–87. IEEE.
- Floyd, R. W. (1962). Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345.
- Lipmaa, H. (2010). On the cca1-security of elgamal and damgård's elgamal. In *International Conference on Information Security and Cryptology*, pages 18–35. Springer.
- Meng, X., Kamara, S., Nissim, K., and Kollios, G. (2015). Grecs: Graph encryption for approximate shortest distance queries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 504–517. ACM.
- Naveed, M., Kamara, S., and Wright, C. V. (2015). Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655.
- Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer.
- Peng, Y., Li, H., Cui, J., Zhang, J., Ma, J., and Peng, C. (2017). hope: improved order preserving encryption with the power to homomorphic operations of ciphertexts. *Science China Information Sciences*, 60(6):062101.
- Samanthula, B. K., Rao, F.-Y., Bertino, E., and Yi, X. (2015). Privacy-preserving protocols for shortest path discovery over outsourced encrypted graph data. In *2015 IEEE International Conference on Information Reuse and Integration*, pages 427–434. IEEE.
- Song, D. X., Wagner, D., and Perrig, A. (2000). Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 44–55. IEEE.

<sup>3</sup><https://www.cit.fraunhofer.de>