Cryptanalysis of a Protocol for Efficient Sorting on SHE Encrypted Data *

Shyam Murthy[0000−0002−0222−322X] and Srinivas Vivek[0000−0002−8426−0859]

IIIT Bangalore, IN
shyam.sm@iiitb.org, srinivas.vivek@iiitb.ac.in

Abstract. Sorting on encrypted data using Somewhat Homomorphic Encryption (SHE) schemes is currently inefficient in practice when the number of elements to be sorted is very large. Hence alternate protocols that can efficiently perform computation and sorting on encrypted data is of interest. Recently, Kesarwani et al. (EDBT 2018) proposed a protocol for efficient sorting on data encrypted using an SHE scheme in a model where one of the two non-colluding servers is holding the decryption key. The encrypted data to be sorted is transformed homomorphically by the first server using a randomly chosen monotonic polynomial with possibly large coefficients, and then the non-colluding server holding the decryption key decrypts, sorts, and conveys back the sorted order to the first server without learning the actual values except possibly for the order.

In this work we demonstrate an attack on the above protocol that allows the non-colluding server holding the decryption key to recover the original plaintext inputs (up to a constant difference). Though our attack runs in time exponential in the size of plaintext inputs and degree of the polynomial but polynomial in the size of coefficients, we show that our attack is feasible for 32-bit inputs, hence accounting for several real world scenarios. Of independent interest is our algorithm for recovering the integer inputs (up to a constant difference) by observing only the integer polynomial outputs.

Keywords: Somewhat Homomorphic Encryption · Comparison · Sorting · Polynomial Reconstruction · Low-depth Circuit

1 Introduction

Cloud hosting solutions that offer pay-as-you-use models provide elasticity and cost-efficiency thus attracting users from varied domains. Cloud providers also offer services and computation capabilities on stored data thereby offloading such overheads from their customers. However, these services can compromise the privacy of the stored data. Hence while data has to be in encrypted form, to be able to make use of the services offered by the cloud, there should be ways to perform meaningful operations on encrypted data. One such service is

* The final publication will be available at www.springerlink.com

to search for k-Nearest Neighbours (k-NN) (according to a given metric) of an encrypted δ-tuple in a database containing n encrypted δ-tuples. k-NN is a basic algorithm used in data mining, machine learning, pattern recognition, etc. Many *efficient* solutions have been proposed for determining k-NN on private data [WCKM09,XLY13,CGLB14,SHSK15,ZHT16], and [ESJ14], [KKN$^+$18] give solutions based on homomorphic encryption schemes.

Secure Sorting and k-NN Protocol from [KKN$^+$18]. Suppose a (possibly very large) data set consists of points in a multi-dimensional vector space with the Euclidean distance as metric and that are stored in encrypted form in the cloud for privacy reasons by a client. Also suppose that the client wishes the server to compute k-NN on this encrypted data by providing an encrypted query point. One obvious approach is to use Fully/Somewhat Homomorphic Encryption (F/SHE) schemes [Gen09, BGV12, GSW13, CGGI19] to perform the computing of the Euclidean distances, sorting and then the computing of the indices of the k-NNs on the encrypted data. But with the current F/SHE schemes it is impractical to even handle data that merely consists of a few hundred elements [ÇDSS15, CS15, ÇS19].

At EDBT 2018, Kesarwani et al. [KKN$^+$18] proposed a secure way to solve the k-NN problem on SHE encrypted data in a model where there is a non-colluding pair of Cloud A and B, *a.k.a.* the federated cloud setting. In this setting, the participating clouds do not collaborate with each other. A client uses Cloud A as storage to store n data points encoded as integer values with each of the δ coordinates encrypted in separate ciphertexts. A user (querier) provides an encrypted query point in a similar format. Server A homomorphically computes the square of the Euclidean distances between the query and the data points in n different ciphertexts using an SHE scheme. The result of the computation is also in encrypted form. Once the distances to the given query point are computed as n ciphertexts, the Server A homomorphically evaluates a monotonic polynomial p of degree d having positive integer coefficients, randomly permutes the order of the ciphertexts and sends them to the Server B. The Server B has access to the full decryption key, who decrypts the received data and sorts the (transformed) plaintext distances, computes the indices of the k-NNs and sends back the indices to Server A which then maps them back to the original encrypted indices and sends the same to the client. The authors of [KKN$^+$18] demonstrate that their method takes only a few minutes when the number of elements is as large as 200,000 and the dimension is 2. It is also claimed in [KKN$^+$18][Section 4.2] that the Server B will not learn anything other than the value of k and the number of equidistant points from the query point. Moreover, the authors claim that if the size of squared plaintext distances is 16 bits, then a polynomial of degree $d = 9$ suffices to ensure that an adversary will only be able to recover the plaintext distances with probability as small as 2^{-160}.

It may be noted that though the protocol of [KKN$^+$18] has been described specifically in the context of securely evaluating k-NN, their technique of transforming inputs through a random monotonic polynomial has applications in

many settings where sorting of SHE encrypted data is needed. Moreover, this protocol may be of interest in scenarios where both computation and then sorting on encrypted data is needed. It may be noted that if sorting is the only functionality required, then order-preserving or order-revealing encryption schemes would suffice for the purpose [BCLO09, BLR+15].

Polynomial Recovery from Only the Outputs. As is evident from the above description of the k-NN (and the sorting) protocol from [KKN+18] on encrypted data, one way of formalizing the cryptanalysis of this protocol is to formulate it as the problem of recovering inputs of a randomly chosen monotonic polynomial with positive coefficients by observing *only* the corresponding outputs. Here the adversary is the Server B who is keen to learn more about the transformed input distances than just their ordering. Formally, let $p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_d \cdot x^d$ be a polynomial of degree d, where each of the integer coefficients a_i is picked uniform randomly and independently in the range $[1, 2^\alpha - 1]$. The polynomial is evaluated (homomorphically) on the (encrypted) unknown n integer inputs $x_i \in [0, 2^\beta - 1]$ $(i = 1, 2, \ldots, n)$. The adversary is provided only n outputs $p(x_i)$. It may be assumed that it knows the parameters d, α and β as assumed in [KKN+18]. The goal is to recover the inputs x_i. In the context of Secure k-NN problem, recovering x_i would correspond to recovering the squares of the Euclidean distances between the query point and data set points.

The problem of polynomial reconstruction, posed in different flavours, has received good attention in the past. A well-known technique for this is the Lagrange interpolation. The problem of polynomial reconstruction also occurs in the context of decoding error-correcting codes with many well-known techniques to recover polynomials even when a sufficiently small fraction of the input-output pairs are error prone [Ber68, GS99, GRS00], and many follow up works. But we would like to emphasize that, to the best of our knowledge, in all the previous works both the input to and the output of the polynomials are given. But in the present setting, only the outputs are provided and we are *not* provided the inputs (except that we only know the range where the inputs come and the degree of polynomial). The goal is to recover the inputs and, consequently, the polynomial itself.

We observe that given only the polynomial outputs there may be many polynomial/input combinations (in the given input range) that result in the same outputs. This is because if $p(x)$ and x_i are the chosen polynomial and the n integer inputs, respectively, then, any polynomial of the form $p(x + c)$ (c, a constant) will result in the same outputs for $x_i - c$ provided all the $x_i - c$ lie in the given interval. So the best we could hope to recover for the current problem is to recover the inputs up to a constant difference. Of course, there are other possibilities too and the number of such equivalent solutions will likely be significantly small if the number of outputs is much larger than the degree of the polynomial. This is indeed the case for the secure k-NN problem when the input data set is very large.

Our Contribution. We give an algorithm (cf. Algorithm 1) to the above defined polynomial reconstruction problem where the goal is to recover the inputs (up to a constant difference) of the randomly chosen monotonic polynomial with positive integer coefficients by observing only their outputs assuming the number of evaluation points is much greater in number compared to the degree of the polynomial. Once $(d+1)$ inputs are recovered, the degree d *integer* polynomial can be reconstructed using the Lagrange interpolation technique. This result invalidates the security claim in [KKN$^+$18][Theorem 4.2] regarding the leakage profile for Server B. In particular, the Server B will be able to learn the square of the Euclidean distances (up to a constant difference) between the query point and the data set points. It may not be able to tell the exact distance to a given point due to random re-ordering but will be able to know all such values. Such an information can potentially help the adversary to narrow down further if it has access to extra information about the underlying data set or query point.

There can be many solutions to the above polynomial reconstruction problem, and hence we will output one solution that satisfies all the output points, (there is also a possibility to enumerate all the solutions). But as discussed above, when the number of output values is far bigger than the input degree, the number of equivalent solutions will likely be small. Our algorithm readily extends to recovering any integer polynomial (not necessarily a monotonic integer polynomial) and any input range (not necessarily $[0, 2^\beta - 1]$). The proposed algorithm (heuristically) runs in time exponential in the size of the inputs (β) and degree (d) of the chosen polynomial, but *polynomially* dependent on the size of the random coefficients (α). We would like to note that in many real world scenarios the inputs are/can be encoded as integers of 16- or 32-bits length and our method is feasible for inputs of such size. Note also that in SHE applications d is required to be not large as well. This is because bigger values of d imply deeper circuits w.r.t. homomorphic multiplications and hence more slow. For the concrete parameters suggested in [KKN$^+$18][Section 4.2], i.e., $\beta = 16$ and $d = 9$, we can recover the inputs (up to a constant difference) for $\alpha = 16$ in a few seconds. We tested our attack using real-world data as well as with uniform random data chosen within the input range, and this is described in detail in Section 3.

Lastly, we investigate in Section 4 another variant of the protocol of [KKN$^+$18] where the (homomorphically) transformed polynomial outputs are perturbed by a noise, yet maintaining the monotonicity. In this case our previously mentioned attack will not work. But we show that it is still possible to recover ratios of the inputs.

2 Cryptanalysis of a Secure *k*-NN Protocol

In Section 2.1 we describe the Secure k-Nearest Neighbour protocol from [KKN$^+$18] and formulate our attack as the polynomial reconstruction problem given only the outputs. We describe our method for polynomial reconstruction and attack on the k-NN protocol in Section 2.2. In Section 2.3 we provide a heuristic run-

ning time analysis of our method. This is followed by an optimisation of our attack in Section 2.4.

2.1 Protocol from [KKN$^+$18]

The Secure k-Nearest Neighbour (Secure k-NN) protocol from [KKN$^+$18] makes use of a non-colluding federated two cloud setting (Figure 1). The data owner outsources his/her database in an encrypted form using an SHE scheme for storage in the Cloud A, whereby the cloud is not privy to the data, thus preserving confidentiality of the data. Each of the n data points are of δ-dimensions. Cloud A provides storage for the database and provides services on the encrypted database homomorphically. One of these services is computation of k-NN of a given δ-dimension query point. End users are clients who are trusted entities for accessing the database and hence possess the secret decryption key shared by the data owner. These users who wish to avail of the computation services provided by Server A form a δ-dimensional query point q, encrypts the same and provides it to Server A.

When Server A receives the query, it homomorphically computes the square of the Euclidean Distances (ED) between the δ-dimensional query point and each of the n data points; if the query point is of the form $q = (q_1, q_2, \ldots, q_\delta)$ and the k^{th} data point is of the form $(k_1, k_2, \ldots, k_\delta)$, then the Euclidean distance between this k^{th} data point and q is $ED_k^2 = (q_1 - k_1)^2 + (q_2 - k_2)^2 + \ldots (q_\delta - k_\delta)^2$. This needs to be computed for each of the n data points in an encrypted manner. Because this ED computation is of multiplicative depth 1, it can be efficiently evaluated using an SHE scheme. The plaintext data points and the query point are encoded as tuples of integers. Note that in the context of F/SHE schemes, fixed-point values too are (exactly) encoded using essentially the scaled-integer representation [CSVW16]. Since the Server A does not possess the decryption key, it will not be able to efficiently uncover the underlying plaintext information of either the query point or the data points. It now picks a monotonic polynomial $p(x)$ of degree d of the form $a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_d \cdot x^d$ for some chosen $d \in \mathbb{N}$, where each of the *integer* coefficients a_i are picked uniform randomly and independently in the range $[1, 2^\alpha - 1]$, for example in the range $[1, 2^{32} - 1]$ as done in [KKN$^+$18][Section 3.4]. This polynomial is then evaluated homomorphically for each of the Euclidean distances and the output ciphertexts are re-ordered using a permutation σ picked uniformly at random, before sending them to Server B for sorting.

Server B possesses the decryption key using which it will decrypt the values received from Server A and sorts them. As the decrypted values are outputs of a random polynomial, the original distances as computed by Server A are "hidden" from Server B. Server B would then send the indices of k-NNs to Server A which would then apply σ^{-1} to the received ordering of the indices and forward the same to the client. The client would decrypt the encrypted indices of the k-NNs of the query point q.

The Server B (and also A) is assumed to be honest but curious. It will perform the computations correctly but is keen to learn more about the distances

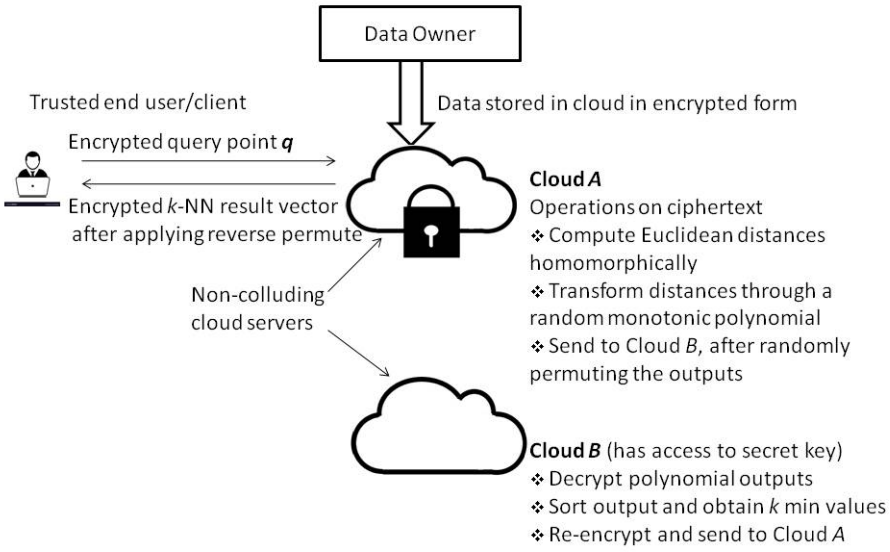

Fig. 1. Secure k-NN Setting from [KKN$^+$18]

between the query point and the data set points. After decryption, the Server B would observe only the outputs of the polynomial evaluation (and *not* the input squared distances). That is, it only sees the values on the L.H.S. of the following set of equations:

$$\begin{aligned} p(x_1) &= a_0 + a_1 \cdot x_1 + a_2 \cdot x_1^2 + \ldots + a_d \cdot x_1^d \\ p(x_2) &= a_0 + a_1 \cdot x_2 + a_2 \cdot x_2^2 + \ldots + a_d \cdot x_2^d \\ &\vdots \\ p(x_n) &= a_0 + a_1 \cdot x_n + a_2 \cdot x_n^2 + \ldots + a_d \cdot x_n^d \end{aligned} \tag{1}$$

It is assumed that the adversary B knows the degree d as it is usually small since homomorphic evaluation of polynomials in encrypted form are efficient only for small degree. It also knows the range $[1, \ldots, 2^\alpha - 1]$ for the unknown coefficients a_i, and the range $[0, \ldots, 2^\beta - 1]$ for the unknown inputs x_i. For our attack, we need not know the exact values for the above three parameters, just an upper bound on them would suffice. Also note that all the parameters above take non-negative integer values.

As noted before, Server A evaluates the polynomial $p(x)$ at n (squared Euclidean distance) integer values $x_1, \ldots, x_n$, and we can assume without loss of

generality that $0 \leq x_1 \leq x_2 \leq \ldots \leq x_n$. Since $p(x)$ is monotonic the ordering of $p(x_i)$ is identical to the ordering of x_i (except possibly when there is equality). If $a_0 \leq x_1$, then for any given tuple of coefficients $(a_0, a_1, \ldots, a_d)$, there will be a set of positive real roots $(\chi_1, \chi_2, \ldots, \chi_d)$ to (1). Hence the authors of [KKN$^+$18] seem to argue that if the range for a_i is large enough, then it will be infeasible to search for all possible polynomials satisfying (1). The authors claim that the probability that Server B successfully recovers the coefficients a_i, followed by x_i, is approximately $1/2^{\alpha \cdot (d+1)}$, which is negligible when α is large. Referring to the example given in [KKN$^+$18, Section 4.2], for $\alpha = 16$ and $d = 9$, this probability is approximately 2^{-160}, which is negligible. Hence the protocol leaks only negligible amount of information about either a_i or x_i to Server B and nothing else other than the order of x_i. We note here that the x_i values may never be uniquely recovered in (1) with probability $= 1$ since $p(x+c)$ is also an equivalent polynomial satisfying the equation for $c \in \mathbb{Z}$ and there may be many values of c such that $0 \leq x_i - c < 2^\beta$. Hence the inputs and the polynomial may only be recovered up to a constant difference. Other non-linear transformations may also result in an equivalent solution. For instance, $p(\sqrt{x})$ can be a potential solution when all the x_i are perfect squares and $p(x)$ contains only even powers. But these possibilities will likely be significantly small when $n \gg d$, which indeed is the scenario in [KKN$^+$18].

2.2 Our Attack

Our key idea is to dramatically reduce the search space of x_i by using the fact that all the roots should be non-negative integers, not just non-negative real values. The pseudocode of our method is given in Algorithm 1 on Page 10 and is described in the steps below.

Step 1 - Guess the differences $(x_i - x_j)$**:** Consider the two equations from (1) for $p(x_i)$ and $p(x_j)$, where $i > j$:

$$p(x_i) - p(x_j) = (x_i - x_j)(\cdot). \tag{2}$$

Let $L_{i,j} = (p(x_i) - p(x_j)) \geq 0$ (as $p(x_i) \geq p(x_j)$), and $D_{i,j}$ be the set of positive divisors of $L_{i,j}$ that are less than 2^β. From (2) we have that $(x_i - x_j)$ is a "small" divisor of $L_{i,j}$. Note that $0 \leq |x_i - x_j| < 2^\beta$. So we can *sieve* all the divisors of $L_{i,j}$ of value less than 2^β. In the sieve method, the quotient of $L_{i,j}/(x_i - x_j)$ is divided by its smallest prime factor and the process of dividing the quotient by its smallest prime factor is continued until we get 1. This is where we crucially use the fact that the inputs and the outputs are represented as integers, and that the (plaintext) input space is small enough to enumerate. The list $D_{i,j}$ constitutes the guesses for the differences of the (unknown but to be determined) values x_i. It turns out that for many values of $L_{i,j}$ there may be too many divisors that are less than 2^β, so we need to sample larger number of output values (i.e., larger n) and carefully pick up d number of $L_{i,j}$'s such that the value of each $D_{i,j}$ is a small positive integer (say, $\leq \psi$) whereby the search space for the guesses

becomes feasible to enumerate. There is another condition on how we choose the set of d many $L_{i,j}$. Namely, we must be able to determine the required $d+1$ many x_i from the given guesses for the differences when one of the free variable, say, x_1 is assigned a value. In other words, the corresponding equations must be linearly independent. Because of the existence of one free variable, the input values can only be determined up to a constant difference. Hence we may assign $x_1 = 0$ if the coefficients of the resulting polynomial are within the given range and this polynomial is consistent with the remaining output values.

Each of the $D_{i,j}$ sets can be visualized as entries of a lower triangular matrix with element $D[i][j]$ represented by the set $D_{i,j}$. One way to determine the d independent set of equations is to stick on to elements of Column 1 of the D matrix for simplicity. We walk the elements of the matrix examining the number of divisors of each of the $D_{i,1}$ lists. As soon as $D_{i,1}$ is greater than ψ, we discard the elements in the particular row $D[i]$. This is continued until we get at least d valid rows in the D matrix. We again note here that the i^{th} row element of Column 1 of D matrix contains sets of divisors for each of $(p(x_i) - p(x_1))$ for $2 \leq i \leq n$ respectively. In Step 2 we look for a consistent set of divisors from Column 1 and in Step 3 we use these d guesses of x_i together with x_1 to compute the required degree d polynomial using Lagrange interpolation and check whether this polynomial is consistent with the remaining $n-d-1$ output values.

Step 2 - Consistency check of the guessed differences: In Step 1 we would have obtained d rows of divisors of polynomial differences in the D matrix that contains the set of guesses for each of the d differences among the unknown inputs. In this step we try to filter out as many guessed tuples as possible before executing the Step 3. This is because the Step 3 below consists of performing Lagrange interpolation and then checking the validity of the constructed polynomial on the remaining inputs and these steps are quite expensive to perform for all the guessed tuples. We iterate over every d-tuple of divisors/guesses in Column 1 of the D matrix and examine to check if that guessed divisor is *consistent* as explained below. For integers i, j with $D_{i,1}$, $D_{j,1}$ representing the set of divisors in Rows i and j in Column 1 of D and given divisors $d_i \in D_{i,1}$ and $d_j \in D_{j,1}$, then d_i and d_j are said to be *consistent* if $(d_i - d_j) \in D_{i,j}$. This is so because if $(x_i - x_1)$ is a divisor of $L_{i,1}$ and $(x_j - x_1)$ is a divisor of $L_{j,1}$ then $(x_i - x_j)$ must be present in $D_{i,j}$, which is evident in the way $L_{i,j}$ is obtained from (2). Only the consistent values are considered and copied to an array $state[]$. In summary, the output of this step is the array $state$ where each of its elements is a consistent divisor of $L_{j,1}$ obtained as above, and it is consistent with every value of $state[j], i \neq j$.

Step 3 - Find a probable polynomial and verify its suitability: Lagrange interpolation using the $(d+1)$ number of (x, y) tuples is used to compute a degree d polynomial. The x values are based on values found in Step 2 and the corresponding y values are the corresponding polynomial outputs, such that if x_i is a divisor of $L_{i,1}$ then y_i is the i^{th} polynomial output enumerated in order.

Since the x values are guesses based on differences of d values of $(x_i - x_1)$, the polynomial obtained by setting x_1 to 0 can actually be a potential polynomial solution. If the polynomial coefficients do not happen to lie within [1,2^α), we can iterate over successive integer values of x from 0 up to $(2^\beta - 1)$, using it as an offset for each of the $(d+1)$ values of x to get solutions that indeed satify the needed bounds. Once a candidate polynomial is identified, using the remaining $(n-d-1)$ points we verify the correctness by computing the roots of this polynomial and checking if they are all in the range $[0, 2^\beta)$. If these verification steps are successful, then the algorithm outputs the coefficients of a polynomial that takes the same values as those of the n input points, and these points are unique up to a constant difference. In other words, the differences of the Euclidean distances are thus recovered in the most likely scenario.

2.3 Running Time Analysis

It looks difficult to do a tight analysis as we need to know the distribution of the divisors of the polynomial outputs evaluated at independent and uniformly random inputs. Hence we only provide a heuristic bound on the expected running time.

Our method makes use of the sieving method to find divisors, in the given range, of the polynomial differences. We then use Lagrange interpolation over $(d+1)$ points to find a polynomial, and then find roots for $c = (n-d-1)$ polynomials that satisfy the validity checks mentioned in Step 3 of Algorithm 1. While the first polynomial output by Lagrange interpolation is very likely the candidate polynomial, we may need to iterate over more values of x_1 (bounded by 2^β) until the polynomial coefficients lie in the range [1,2^α) and all the x_i are in [0,2^β).

Suppose we consider ψ as a small integer bound on the number of divisors for each of the d polynomial differences. Then the bound on the size of the search space for the divisors is ψ^d. The number of divisors of an integer N is bounded by $N^{O(\frac{1}{\log\log N})}$; and on the average case it is $\log N$ [Tao08]. Though we only consider divisors bounded by 2^β, for ease of analysis we use the $\log N$ expected bound. Based on this, we can set the value of ψ to be approximately equal to $\alpha + d\beta$, whereby the expected value of search space size is $O(\alpha + d\beta)^d$. We then find the consistent set of divisors as described in Step 2 of Algorithm 1 and the worst case scenario to assume is that all the d-tuple divisors/guesses are consistent. Using each of the ψ^d many d-tuple divisors/guesses, in the worst case, we need to iterate over 2^β values of x_1 doing a Lagrange interpolation for $(d+1)$ points and root finding for c polynomials. With Lagrange interpolation being a $O(d^2 \cdot (\alpha + d\beta)^2)$ algorithm and so is the cost of a root finding, then the total cost for Step 3 comes to $\tilde{O}(\alpha + d\beta)^d \cdot 2^\beta \cdot n)$.

The heuristic expected running time of Algorithm 1 is $\tilde{O}((\alpha + d\beta)^d \cdot 2^\beta \cdot n)$.

Algorithm 1: Integer Polynomial Reconstruction From Only the Outputs

Procedure `Main(`*Polynomial outputs :* $\{p(x_1), \ldots, p(x_n)\}$`)` **:**
- D = GuessTheDifference($\{p(x_1), \ldots, p(x_n)\}, \psi$)
- *state* = CheckConsistency(D)
- Q = FindCandidatePolynomial(*state*, $\{p(x_1), \ldots, p(x_n)\}$)
- **return** Q

Procedure `GuessTheDifference(`*Polynomial outputs :* $\{p(x_1), \ldots, p(x_n)\}$, ψ`)` **:**
- **for** $i = 2$ **to** (n) **do**
  - **for** $j = 1$ **to** $(i-1)$ **do**
    - Use the sieve method to obtain all the (positive) divisors less than 2^β of $L_{i,j} = (p(x_i) - p(x_j))$ /* $p(x_i) > p(x_j)$ */
    - $D_{i,j}$:= Set of all divisors of $L_{i,j}$ less than 2^β
  - **end**
- **end**
- /* We now have D: a lower triangular matrix */
- *valid_row_count* = 0 /* Count rows in which all row elements have their divisor count less than ψ */
- **forall** D_i **do**
  - **forall** $D_{i,j}$ elements in D_i **do**
    - **if** Sizeof($D_{i,j}$) $> \psi$ **then**
      - /* Number of factors of $D_{i,j}$ more than threshold */
      - Discard row D_i /* Mark $D_{i,1}$ as -1 */
      - Break out of this loop and start enumeration on row D_{i+1}
    - **end**
  - **end**
  - *valid_row_count*++
  - **if** *valid_row_count* == d **then**
    - /* We now have d valid rows in D matrix */
    - **break** /* Out of outer loop */
  - **end**
- **end**
- Compact D matrix by removing all rows having first element = -1.
- /* First row of D contains 0s, next $(d+1)$ rows contain valid values */
- **return** D /* Set of divisors matrix */

Procedure `CheckConsistency(`*Set of positive divisors matrix* D`)` **:**
- **for** $i = 3$ **to** $(d+2)$ **do**
  - **for** $j = (i-1)$ **downto** 2 **do**
    - $\forall (d_i, d_j)$ where $d_i \in D_{i,1}$ and $d_j \in D_{j,1}$ and $d_i \neq 0$, $d_j \neq 0$
    - **if** $(d_i - d_j) \notin D_{i,j}$ **then**
      - Set $d_i = 0$ in $D_{i,1}$
    - **end**
  - **end**
- **end**
- **forall** $d_j \neq 0 \in D_{2,1}$ **do**
  - **for** $i = 3$ **to** $(d+2)$ **do**
    - **if** $d_i \neq 0$ and $(d_i - d_j) \notin D_{i,1}$ **then**
      - Set $d_j = 0$ in $D_{2,1}$
    - **end**
  - **end**
- **end**
- Iterate over D_i and populate *state*[i] with non-zero divisor of $D_{i,0}$ where i is suitably offset to populate *state*[] starting from index 0
- **return** *state*[] /* Divisor set consistent over all elements $\in L$ */

/* Continued on next page */

Procedure `FindCandidatePolynomial(`*Consistent Divisor set* $\{state\}$, *Polynomial outputs* $\{p(x_1), \ldots, p(x_n)\}$`)` **:**

for $\nu = 0$ **to** $(2^\beta - 1)$ **do**
  for $i = 0$ **to** d **do**
    Form a set of tuples G := $\{(a, b) : a = (\nu + state[i])$ and $b = p(x_i)\}$
  end
  Use Lagrange interpolation on G to get polynomial Q
  Verify if $1 \leq$ coefficient of $Q < 2^\alpha$ is true for all coefficients of Q
  Verify if Q has non-negative integer roots, which are $< 2^\beta$, with respect to the remaining $(n - d - 1)$ polynomial outputs, namely, $p(x_{d+2})$ to $p(x_n)$.
  If all the above verification steps are successful, **return** Q
end
return 0

2.4 Further Optimisation

In Algorithm 1, in the worst case, we need to enumerate over ψ^d many d-tuples of divisors/guesses. We provide here a way to choose the divisor sets such that the enumeration complexity is as small as possible.

We refer to (1) giving the polynomial outputs. We compute differences $p(x_i) - p(x_j)$, where $p(x_i) \geq p(x_j)$ if $x_i \geq x_j$. These ${}^nC_2$ values can be represented as lower a triangular matrix L, with elements $L_{i,j}$ as described in Step 1 of Algorithm 1, is given below :

$$L = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ p(x_2) - p(x_1) & 0 & 0 & \ldots & 0 \\ p(x_3) - p(x_1) & p(x_3) - p(x_2) & 0 & \ldots & 0 \\ \vdots & & & & \\ p(x_n) - p(x_1) & p(x_n) - p(x_2) & p(x_n) - p(x_3) & \ldots & p(x_n) - p(x_{n-1}) \quad 0 \end{pmatrix}$$

The number of divisors for each of the differences up to a bound ψ are obtained by sieving, represented as a matrix D where the element D_{ij} represents the set of divisors (with values $< 2^\beta$) of $(p(x_i) - p(x_j))$.

$$D = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ D_{2,1} & 0 & 0 & \ldots & 0 \\ D_{3,1} & D_{3,2} & 0 & \ldots & 0 \\ \vdots & & & & \\ D_{n,1} & D_{n,2} & D_{n,3} & \ldots & D_{n,n-1} \quad 0 \end{pmatrix}$$

We now need to find the set of d many D_{ij} elements of the matrix D such that $\prod D_{ij}$ is minimum, in other words, the product of the number of guesses is minimum. This set of elements in the matrix D can be visualized as a complete

undirected graph on n vertices wherein the number of elements in D_{ij} is the edge cost between nodes V_i and V_j. Now finding the minimum $\prod D_{ij}$ is akin to finding the d-Minimum Spanning Tree (i.e., a minimum weight tree with d edges only) in the graph, where the weight of the tree is represented by the product of the weights. The requirement that the subgraph is a tree comes from the linear independence requirement of the corresponding set of equations. Essentially, we are transforming the problem of finding the small search space of divisors to the problem of finding a d-minimum spanning tree having the least cost (in terms of divisor product) across all divisors sets of matrix D yet satisfying the linear independence condition. It is shown in [RSM+96] that the d-MST problem is NP-hard for points in the Euclidean plane. The same paper provides an approximation algorithm to find d-MST with performance ratio of $2\sqrt{d}$ for a general edge-weighted graph, with non-negative edge weights. Note that this approximation algorithm also works for multiplication of edge weights (weights greater than 1) since by extraction of logarithms this can be trivially turned into addition of edge weights. Using this algorithm, we can carefully select $d < n$ nodes having close to the minimum enumeration complexity. in order to make our search space feasible to guess the differences $(x_i - x_j)$ with $x_i \geq x_j$. From the d-MST so obtained, we can now go on to find the set of divisors $(x_i - x_j)$ such that they are consistent as explained in Section 2.2 and continue with finding the polynomial coefficients using Lagrange interpolation as described in Algorithm 1. However, we did not implement this optimisation in our code as the concrete running time was already small enough. But for larger instances this optimisation will be useful.

3 Experiments and Results

We have performed two sets of experiments using the SAGE library [The19]. Our source code is available at [MV]. One set consists of choosing random values within given bounds for x_i and a_i, computing the outputs of a degree d polynomial, and trying to recover the coefficients of the polynomial, namely the a_i values using only the polynomial output values. The second set consists of using data available from the UCI Machine learning repository [DG17] which is a real-world hospital data obtained from a hospital in Caracas, Venezuela.

All our experiments were run on a Lenovo ThinkStation P920 workstation having a 2.3 GHz Intel®Xeon® processor with 12 cores. The algorithms for sieving, consistency check and polynomial verification were exactly same in both the cases, the only difference being in the datasets as described in the respective sections below. As in [KKN+18], we have chosen the degree of the polynomial $d = 9$.

3.1 Experiments with Random Values

We set the bound for a_i and x_i as given in Table 1 with the values being uniform random and independently chosen from the respective ranges. We computed the

polynomial with a_i as coefficients and computed $n = 40$ outputs for the x_i values. These n values were the input to our algorithm. The choice of $n = 40$ was based on observations from the experiments; in majority of the instances we could bound the number of divisors to less than 20 thereby making the search space significantly less than 20^{10}. We then used the divisor set and $(d+1)$ polynomial outputs to compute a possible polynomial using Lagrange interpolation which we then used to verify successfully against the remaining $(n - d - 1)$ output values. We note that our search space is significantly less than the estimate of 2^{160} in [KKN+18]. It looks like many further optimization could be done to reduce the search space. When we increased β to 32 for x_i, SAGE encountered an out-of-memory error while performing Lagrange interpolation. But we think it should still be feasible to run this instance too.

α (in bits)	β (in bits)	Run time (in seconds)
16	16	4
16	24	288
16	28	552
24	16	8
24	24	374
24	28	1283
32	16	9
32	24	241
32	28	1676

Table 1. Run times for polynomial reconstruction for random parameters.

3.2 Experiments with Real World Data

We used the cervical cancer (risk factors) data set, same as the one used by [KKN+18], also available from the UCI Machine learning repository [DG17]. This data set consists of information pertaining to 858 patients, each consisting of 32 attributes comprising of demographic information, habits and historic medical records. The dataset had a few missing values due to privacy concerns and these were set to 0. Values with fractional part were rounded off to the nearest integer. We repeated the experiment with different random polynomials and were able to recover the polynomial successfully up to the differences. We also tested with 16, 24 and 32 bit values of α and have tabulated the time taken by SAGE to compute the polynomial in each of the cases. $\beta = 16$ was suffice to encode this data. Time for execution is given seconds and is averaged over 5 runs in each case.

Our results invalidate the security claims in [KKN+18][Theorem 4.2] regarding the leakage profile for Server B. For the parameters mentioned in [KKN+18]

α (in bits)	β (in bits)	Run time (in seconds)
16	16	2.25
24	16	73.81
32	16	109.87

Table 2. Run times for polynomial reconstruction for a real world data.

[Section 4.2], i.e., $d = 9$ and the (squared plaintext) distances are in the range $[0, 2^\beta)$, where, $\beta = 16$. For the parameters mentioned there, with only $n = 40$ output values, we could recover the coefficients of the polynomial (up to a constant difference) within a few minutes as given in Table 2.

Because of the random re-ordering of the distances, Server B will not learn the exact distance of the query point to a specified point (say the i^{th} point in the original order). Nevertheless, in many real world scenarios the data set is publicly available and this, and perhaps other auxilliary information, may potentially be used in combination with our results to leak information about the query point.

4 Attack on the Secure k-NN protocol in the Noisy Setting

In this section we give another attack on the protocol of [KKN$^+$18] if one tries to overcome our attack from Section 2 by perturbing the polynomial outputs by adding noisy error terms. This modified protocol is not mentioned in [KKN$^+$18] but we consider it here for completeness.

In the original solution given in [KKN$^+$18], in order to hide the Euclidean distance values, Server A chooses a monotonic polynomial and homomorphically evaluates this polynomial on its computed distances and permutes the order before sending them to Server B. Now, instead of sending these (encrypted) polynomial outputs as it is, if they are perturbed with some noise such that the ordering is still maintained, it will make our attack in Section 2 unsuccessful in recovering the polynomial or the inputs, as the attack relies on the exact difference of the polynomial outputs. It is easy to see that the error value can only be as large as the sum of all the coefficients except the constant term. Let $P(x) = a_0 + a_1 \cdot x + \ldots + a_d \cdot x^d$ be the chosen monotonic polynomial, then, $P(0) = a_0$, $P(1) = a_0 + a_1 + \ldots + a_d$ and the maximum value of the added noise needs to be less than $(P(1) - P(0))$ so as to maintain the original ordering of polynomial outputs, meaning the perturbation error may only be chosen from the set $[0, 1, \ldots, (a_1 + \ldots + a_d)]$. This safe choice of the error term is due to the fact that the polynomial output values are encrypted and hence it is not possible for the Server A to inspect the value and accordingly choose the error term. The range of perturbation error terms still depends on the size of the coefficient space that can potentially be very large (unlike the plaintext space as assumed). The attack presented in the previous section will not work in this new setting because in the attack we rely on the exact differences of the polynomial outputs.

In this new setting, we next show that it is still possible to leak ratios of the inputs to the Server B, although recovering the exact values (even up to to a constant difference) may be challenging. But still a lot more information about the inputs is leaked than just a single bit. Let two of the values that the Adversary B obtains after decryption be $F(x_i) = P(x_i) + e_i$ and $F(x_j) = P(x_j) + e_j$, where e_i and e_j are the random error terms such that $0 \leq e_i, e_j < \sum_{k=1}^{d} a_k$ and $1 \leq a_k < 2^{\alpha}$. Consider the ratio $F(x_i)/F(x_j)$ with $0 \leq x_j \leq x_i < 2^{\beta}$:

$$\frac{F(x_i)}{F(x_j)} = \frac{(\sum_{k=0}^{d} a_k) + a_1 \cdot x_i + \ldots + a_d \cdot x_i^d}{(\sum_{k=0}^{d} a_k) + a_1 \cdot x_j + \ldots + a_d \cdot x_j^d}. \tag{3}$$

Note that each $F(x_k) > 0$. When x_i and x_j are sufficiently large we obtain that the ratio in (3) is approximately close to $(x_i/x_j)^d$. By taking the d^{th} root of this value, we can recover the ratio (x_i/x_j). Note also that if the error terms e_k were not significantly small than the leading terms (which, fortunately, is *not* the case), then we would not be able to recover the ratios.

5 Conclusion and Future Work

In this paper we give an attack on the protocol of [KKN$^+$18] for Secure k-NN on encrypted data. This attack is based on our algorithm for integer polynomial reconstruction given only the outputs. While, by just using the outputs, it is not possible to accurately determine the coefficients or the inputs, we show that we can feasibly recover the inputs (up to a constant difference) of size about 32 bits when the number of outputs is much bigger than the degree of the polynomial. Our experiments were conducted both on uniformly randomly selected values as well as a real-world dataset. Since many of the datasets are available in the public domain it may possible for an adversary to derive more information about the exact values using our method together with some other available metadata.

Our method for polynomial reconstruction runs in exponential time in plaintext space β and degree d of the chosen polynomial. In many real-world scenarios both these parameters will be small. Future work can look at having a better algorithm and/or have a lower bound analysis of the time required for this polynomial reconstruction problem. Finally, an FHE solution that can perform efficient sorting and searching on large datasets would eliminate the need for service providers to be entrusted with decryption keys, thereby providing a more secure cloud computation environment.

Acknowledgements

We thank Sonata Software Limited, Bengaluru, India for funding this work. We also thank Debdeep Mukhopadhyay and Sikhar Patranabis for helpful discussions. We also thank V.N. Muralidhara for pointing out results on the k-MST problem.

References

BCLO09. Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-Preserving Symmetric Encryption. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 224–241, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

Ber68. ER Berlekamp. Algebraic Coding Theory, McGraw-Hill. *New York*, Vol8, 1968.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

BLR$^+$15. Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically Secure Order-Revealing Encryption: Multi-input Functional Encryption Without Obfuscation. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 563–594, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

ÇDSS15. Gizem S. Çetin, Yarkin Doröz, Berk Sunar, and Erkay Savas. Depth Optimized Efficient Homomorphic Sorting. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America*, volume 9230 of *Lecture Notes in Computer Science*, pages 61–80, Guadalajara, Mexico, August 23–26, 2015. Springer, Heidelberg, Germany.

CGGI19. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology*, Apr 2019.

CGLB14. Sunoh Choi, Gabriel Ghinita, Hyo-Sang Lim, and Elisa Bertino. Secure kNN Query Processing in Untrusted Cloud Environments. *IEEE Transactions on Knowledge and Data Engineering*, 26:2818–2831, 2014.

CS15. Ayantika Chatterjee and Indranil Sengupta. Searching and Sorting of Fully Homomorphic Encrypted Data on cloud. *IACR Cryptology ePrint Archive*, 2015:981, 2015.

ÇS19. Gizem S. Çetin and Berk Sunar. Homomorphic Rank Sort Using Surrogate Polynomials. In Tanja Lange and Orr Dunkelman, editors, *Progress in Cryptology – LATINCRYPT 2017*, pages 311–326, Cham, 2019. Springer International Publishing.

CSVW16. Anamaria Costache, Nigel P. Smart, Srinivas Vivek, and Adrian Waller. Fixed-point arithmetic in SHE schemes. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016: 23rd Annual International Workshop on Selected Areas in Cryptography*, volume 10532 of *Lecture Notes in Computer Science*, pages 401–422, St. John's, NL, Canada, August 10–12, 2016. Springer, Heidelberg, Germany.

DG17. Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017.

ESJ14. Yousef Elmehdwi, Bharath K. Samanthula, and Wei Jiang. Secure k-Nearest Neighbor Query over Encrypted Data in Outsourced Environments. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 664–675, 2014.

Gen09. Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA, 2009. AAI3382729.

GRS00. O. Goldreich, R. Rubinfeld, and M. Sudan. Learning Polynomials with Queries: The Highly Noisy Case. *SIAM Journal on Discrete Mathematics*, 13(4):535–570, 2000.

GS99. V. Guruswami and M. Sudan. Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, Sep. 1999.

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

KKN+18. Manish Kesarwani, Akshar Kaul, Prasad Naldurg, Sikhar Patranabis, Gagandeep Singh, Sameep Mehta, and Debdeep Mukhopadhyay. Efficient Secure k-Nearest Neighbours over Encrypted Data. In *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018.*, pages 564–575, 2018.

MV. Shyam Murthy and Srinivas Vivek. Available at `http://github.com/shyamsmurthy/knn_polynomial_recovery`. Last accessed on 22nd September, 2019, at 15:30.

RSM+96. R. Ravi, Ravi Sundaram, Madhav V. Marathe, Daniel J. Rosenkrantz, and S. S. Ravi. Spanning Trees - Short or Small. *SIAM J. Discrete Math.*, 9(2):178–200, 1996.

SHSK15. Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. Compacting Privacy-Preserving k-Nearest Neighbor Search using Logic Synthesis. *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015.

Tao08. Terence Tao. Blog: The divisor bound, 2008. Available at `https://terrytao.wordpress.com/2008/09/23/the-divisor-bound/`. Last accessed on 19th July, 2019, at 15:30.

The19. The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.4)*, 2019. `https://www.sagemath.org`.

WCKM09. Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. Secure kNN Computation on Encrypted Databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 139–152, New York, NY, USA, 2009. ACM.

XLY13. Xiaokui Xiao, Feifei Li, and Bin Yao. Secure Nearest Neighbor Revisited. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, pages 733–744, Washington, DC, USA, 2013. IEEE Computer Society.

ZHT16. Youwen Zhu, Zhiqiu Huang, and Tsuyoshi Takagi. Secure and Controllable k-NN Query over Encrypted Cloud Data with Key Confidentiality. *Journal of Parallel and Distributed Computing*, 89(C):1–12, 2016.