SeCCA: Towards Privacy-preserving Biclustering Algorithm with Homomorphic Encryptions

Shokofeh VahidianSadegh$^{1[0000-0002-6464-6842]}$,
Lena Wiese$^{1[0000-0003-3515-9209]}$, and
Michael Brenner$^{2[0000-0003-2690-7340]}$

^{1} Institute of Computer Science
Goethe University Frankfurt
Frankfurt, Germany
`VahidianSadegh@mathematik.uni-frankfurt.de`
`lwiese@cs.uni-frankfurt.de`
^{2} Leibniz Universitaet Hannover
Hannover, Germany
`brenner@luis.uni-hannover.de`

Abstract. Massive amounts of newly generated gene expression data have been used to further enhance personalised health predictions. Machine learning algorithms prepare techniques to explore a group of genes with similar profiles. Biclustering algorithms were proposed to resolve key issues of traditional clustering techniques and are well-adapted to the nature of biological processes. Besides, the concept of genome data access should be socially acceptable for patients since they can then be assured that their data analysis will not be harmful to their privacy and ultimately achieve good outcomes for society [1]. Homomorphic encryption has shown considerable potential in securing complicated machine learning tasks. In this paper, we prove that homomorphic encryption operations can be applied directly on biclustering algorithm (Cheng and Church algorithm) to process gene expression data while keeping private data encrypted. This Secure Cheng and Church algorithm (SeCCA) includes nine steps, each providing encryption for a specific section of the algorithm. Because of the current limitations of homomorphic encryption operations in real applications, only four steps of SeCCA are implemented and tested with adjustable parameters on a real-world data set (yeast cell cycle) and synthetic data collection. As a proof of concept, we compare the result of biclusters from the original Cheng and Church algorithm with SeCCA to clarify the applicability of homomorphic encryption operations in biclustering algorithms. As the first study in this domain, our study demonstrates the feasibility of homomorphic encryption operations in gene expression analysis to achieve privacy-preserving biclustering algorithms.

Keywords: Gene Expression, Biclustering Algorithm, Privacy-Preserving AI, Homomorphic Encryption

1 Introduction

Gene expression data are being generated through high-throughput technologies such as Microarray and RNA-seq. Biclustering algorithms – for instance, Cheng and Church [2] – have been used extensively in this regard for discovering meaningful groups and a link between sets of genes and sample traits [3]. Notwithstanding the importance of this analysis aspect, the rise in availability and use of genomic data raises a growing concern about its security and privacy [4]. Homomorphic encryption with certain operations (additions and/or multiplications) is able to handle sensitive genomic data by allowing data to remain encrypted even during computation. Accordingly, researchers have used the potential of homomorphic encryption operations in machine learning applications – particularly medical data analysis. To the best of our knowledge, existing recent studies have not addressed biclustering algorithms exclusively; this research gap leads to a lack of secure data processing in this area. We intend to realise the possibility of applying homomorphic encryption operations over biclustering algorithms, with our experimental results reported here showing promising results. Our Secure Cheng and Church algorithm (SeCCA) consists of nine different steps, each of which ensures partial security of the algorithm. In practice, due to the limitations of homomorphic operations discussed below, we benchmarked four of them successfully: we tested the computational performance of these four SeCCA steps and compared the result of them with the original algorithm without any encryption/decryption. The distinct contribution of this manuscript is to provide privacy-preserving steps for a biclustering algorithm, specifically Cheng and Church algorithm (SeCCA), as for the first time Cheng and Church introduced biclustering as a new paradigm to simultaneously cluster both genes and conditions. Since then, many other such algorithms have been published but mostly compare their works with this algorithm which has been a foundation and proof-of-concept implementation of biclustering. In this paper, we focus on the secure computation of the mean squared residue score that restricts access to sensitive private data by homomorphic encryption operations. Afterwards, we compare the biclusters obtained by non-encrypted with the ones obtained by the encrypted Cheng and Church algorithm by means of external evaluation measure (clustering error). All code and data sets are publicly available at `https://github.com/ShokofehVS/SeCCA`.

2 Related Work

Before approaching the core of our privacy-preserving procedure, we elaborate on concepts of biclustering algorithms. Further in this section, we mention relevant research on private genome data analysis applying cryptographic techniques.

2.1 Biclustering Algorithms

Traditional clustering algorithms are one-way clustering methods grouping observations according to similarities among all variables at the same time [5]. In addition, some studies elaborate that a biological process may be active only under subsets of genes as well as subsets of samples. There are also some genes or samples that may not participate in any cluster; hence it is of paramount importance to go beyond a traditional clustering prototype and apply a more adapted technique like the biclustering [6] which simultaneously clusters the rows and columns of a matrix for sorting heterogeneous data into homogeneous blocks [7].

2.2 Private Genomic Data Analysis

Numerous privacy-preserving clustering methods based on homomorphic encryption have been proposed; in the following, we survey some of these. The authors in [8] implemented a privacy-preserving evaluation algorithm for support vector clustering. By this model, the cluster label was allocated for new test data without decryption, which improved the clustering performance for non-convex data. A deep neural network with fully homomorphic encryption introduced in [9] adopted approximation methods and used bootstrapping to evaluate an arbitrary deep learning model on encrypted data. A secure K-means (CKKSKM) was developed in [10] to encrypt outsourcing data based on the CKKS scheme to prevent revealing private information. This scheme reduced the overhead of outsourcing data to the cloud for storage and calculation. In some experiments, clustering algorithms are represented with focus on multiple encryption techniques including homomorphic encryption and multiparty computation. [19] introduces the first practical and fully private density-based clustering scheme based on secure two-party computation. A scalable privacy-preserving clustering algorithm in [20] is designed for multi-party clustering in a modular approach which is five orders of magnitude faster than the current solutions. FHE-friendly K-Means-Algorithm on encrypted data by [21] presents a natural encoding that makes division by an encrypted value possible with some restrictions in terms of performance in practice. As opposed to the above approaches, in our paper, we present and analyse proof-of-concept implementations of biclustering (particularly Cheng and Church Algorithm) on homomorphically encrypted gene expression data – a use case for which no prior approaches could be found.

3 Theoretical Background

3.1 Cheng and Church Biclustering Algorithm

Biclustering algorithms – like Cheng and Church [2] – have been used extensively in gene expression analysis which results in a better understanding of diseases like cancer [22]. The concept of bicluster refers to a subset of genes and a subset of conditions with a high similarity score, which measures the coherence of the genes and conditions in the bicluster. It also returns the list of biclusters for the given data set. Important notations in the paper containing input parameters are summarised in Table 1. In addition, “enc” label at the beginning of each symbol like $enc\_maxValue$ indicates an encrypted expression which here is the encrypted maximum value of the data matrix. The residue of a cell in the bicluster (a_{ij}) over the subsets of the rows (I) and the columns (J) is defined by:

$$a_{ij} - a_{iJ} - a_{Ij} + a_{IJ}$$

Where a_{iJ}, a_{Ij}, and a_{IJ} indicate the mean of the ith row, the mean of the jth column and that of all elements in the bicluster respectively. More precisely, a submatrix A_{IJ} is determined by the pair (I, J) where the row means are $a_{iJ} = \frac{1}{|J|}\sum_{j \in J} a_{i,j}$; and similarly, the column means are $a_{Ij} = \frac{1}{|I|}\sum_{i \in I} a_{i,j}$ along with the mean in the submatrix (I, J) as below:

$$a_{IJ} = \frac{1}{|I||J|}\sum_{i \in I, j \in J} a_{ij} = \frac{1}{|I|}\sum_{i \in I} a_{iJ} = \frac{1}{|J|}\sum_{j \in J} a_{Ij}$$

The mean squared residue score can be described as the variance of all elements in the bicluster, the mean of row variance, and the mean of column variance:

$$H(I, J) = \frac{1}{|I||J|}\sum_{i \in I, j \in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2$$

The focus is on finding biclusters having low mean squared residue score and particularly large, maximal ones with scores below a predefined threshold. If $H(I, J) \leq \delta$ for $\delta \geq 0$, a submatrix A_{IJ} is referred to δ-bicluster. The algorithm starts with a large matrix (one with all the data) and then proceeds by removing the row or column to achieve the largest decrease in the score. Accordingly, the computation of the score of all the submatrices is needed after removing any row or column. First, a step for the simultaneous deletion of multiple rows/columns is performed over the input data set; this is followed by a single row/column deletion step (see Algorithms 2 and 3). In the next step, the result of applying node deletion may not be maximal, so some rows and columns can be added without increasing the mean squared residue score (see Algorithm 4). Add that, in the original study, Cheng and Church attempted to find α (a threshold for multiple node deletion) as large as possible and run the algorithm on 100 biclusters in less than 10 minutes as well. Steps 3 and 4 (refer to Section 4.2) won't be applied when the number of conditions is less than 100 for the yeast data set; therefore, only algorithm 1 is called by starting deleting conditions.

3.2 Homomorphic Encryption

Homomorphic Encryption (HE) was coined first by Rivest in 1978 with the concept of "privacy homomorphism". Homomorphic encryption is considered a secure computation method on encrypted data (ciphertext) in which the result of the computation is also ciphertext. By decrypting the result in ciphertext, the decrypted result should be identical to the output of operations on non-encrypted (plaintext) data [11]. We apply Pyfhel as a python wrapper for the

Algorithm 1 Finding a Given Number of Biclusters

```
Require: $A$ Matrix of real numbers with possible missing elements, $\delta \geq 0$ the maximum acceptable mean squared residue score, $\alpha \geq 1$ a parameter for multiple node deletion and $n$ the number of $\delta$-biclusters to be found
Ensure: Replacing missing elements in $A$ with random numbers, $A'$ a copy of matrix $A$
1: Apply multiple node deletion algorithm on $A', \delta$, and $\alpha$ for rows (columns) greater than or equal to 100 (the result matrix $B$)
2: Apply single node deletion algorithm on $B$, and $\delta$ (the result matrix $C$)
3: Apply node addition algorithm on $A$ and $C$ (the result bicluster $D$)
4: Report D, and replace the elements in $A'$ that are also in D with random numbers
```

Algorithm 2 Single Node Deletion

```
Require: $A$ Matrix of real numbers, $\delta \geq 0$ the maximum acceptable mean squared residue score
Ensure: $A_{IJ} = A$ where $I$ and $J$ are initialised to the gene and condition sets in the data
1: Compute $a_{iJ}$ for all $i \in I, a_{Ij}$ for all $j \in J, a_{IJ}$ and $H(I, J)$. If $H(I, J) \leq \delta$ return $A_{IJ}$
2: Find the row $i \in I$ with the largest $d(i) = \frac{1}{|J|}\sum_{j\in J}(a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2$
3: Find the column $j \in J$ with the largest $d(j) = \frac{1}{|I|}\sum_{i\in I}(a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2$
4: Remove the row or column having the larger $d$ value by updating $I$ or $J$
```

Algorithm 3 Multiple Node Deletion

```
Require: $A$ Matrix of real numbers, $\delta \geq 0$ the maximum acceptable mean squared residue score, $\alpha > 1$ a threshold for multiple node deletion
Ensure: $A_{IJ} = A$ where $I$ and $J$ are initialised to the gene and condition sets in the data
1: Compute $a_{iJ}$ for all $i \in I, a_{Ij}$ for all $j \in J, a_{IJ}$ and $H(I, J)$. If $H(I, J) \leq \delta$ return $A_{IJ}$
2: Remove the rows $i \in I$ with $\frac{1}{|J|}\sum_{j\in J}(a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2 > \alpha H(I, J)$
3: Recompute $a_{Ij}, a_{IJ}$ and $H(I, J)$
4: Remove the columns $j \in J$ with $\frac{1}{|I|}\sum_{i\in I}(a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2 > \alpha H(I, J)$
5: If nothing has been removed in the iteration, switch to Algorithm 2
```

Algorithm 4 Node Addition

```
Require: $A$ Matrix of real numbers, $I$ and $J$ signify a $\delta$-bicluster
1: Compute $a_{iJ}$ for all $i$, $a_{Ij}$ for all $j$, $a_{IJ}$ and $H(I, J)$
2: Add the columns $j \notin J$ with $\frac{1}{|I|}\sum_{i\in I}(a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2 \leq H(I, J)$
3: Recompute $a_{iJ}$, $a_{IJ}$ and $H(I, J)$
4: Add the rows $i \notin I$ with $\frac{1}{|J|}\sum_{j\in J}(a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2 \leq H(I, J)$
5: For each row $i$ still not in $I$, add its inverse if $\frac{1}{|J|}\sum_{j\in J}(-a_{ij} + a_{iJ} - a_{Ij} + a_{IJ})^2 \leq H(I, J)$
6: If nothing is added in the iteration, return the final $I$ and $J$ as $I'$ and $J'$
```

Table 1: Notation and Symbols

Symbols	Description
a_{ij}	Element of expression matrix
a_{iJ}	Mean of the ith row
a_{Ij}	Mean of the jth column
a_{IJ}	Mean of all elements
$H(I,J)$	Mean squared residue of submatrix I and J
$\delta \geq 0$	Maximum acceptable mean squared residue score
$\alpha \geq 1$	Parameter for multiple node deletion
n	number of δ-biclusters to be found
enc_a_{ij}	Encrypted element of expression matrix

Microsoft SEAL library in our project because of its convenience to use and higher-level API. We use BFV [13] as a highly efficient fully homomorphic encryption scheme for applications working over larger amounts of data with the following settings that are chosen based on a number of examinations and according to the suggested setting parameters in creating context:

- The plaintext modulus p specifies the prime number applied on the polynomial's coefficients that determines how large encrypted values can get before wrap-around; we set $p = 1964769281$.
- The polynomial modulus m specifies the degree of the irreducible polynomial $x^m + 1$ of the underlying polynomial ring R; this ring is defined in BFV to be a cyclotomic ring $R = \mathbb{Z}[x]/(x^m + 1)$, where m is a power of 2; we set $m = 8192$.
- In addition, we set Pyfhel's security level parameter to 192 bit.
- We set the encoding scheme to fractional; in addition, 64 bits for integer and 4096 bits for the decimal part are adjusted.
- For BFV scheme, q is determined utilising the largest value that achieves 192-bit security for the given polynomial degree (parameter sec) [12]. Consequently, $logq$ is 152 according to the setup and parameters.

4 Secured CCA (SeCCA)

In this section, firstly, we describe a possible message flow for secure processing of the Cheng and Church algorithm consisting of two agents (data owner and storage system). Afterwards, we clarify how a homomorphic encryption scheme could be utilised to ensure the security of gene expression data analysis by biclustering algorithms (particularly the Cheng and Church algorithm).

4.1 Workflow

In our threat model, the gene expression data are privately kept by the data owner. We assume the server to be semi-honest ("honest but curious" [14]); hence, by processing encrypted data we aim to ensure that any data cannot be leaked on the server side. In addition, the biclustering algorithm that we have used (Cheng and Church's) is public and we trust the cloud server to do computations correctly. It is worth mentioning the message flow which is inspired by [15] consisting of two agents such as the data owner (patient) and storage system (cloud) for secure processing in our scenario (Figure 1).

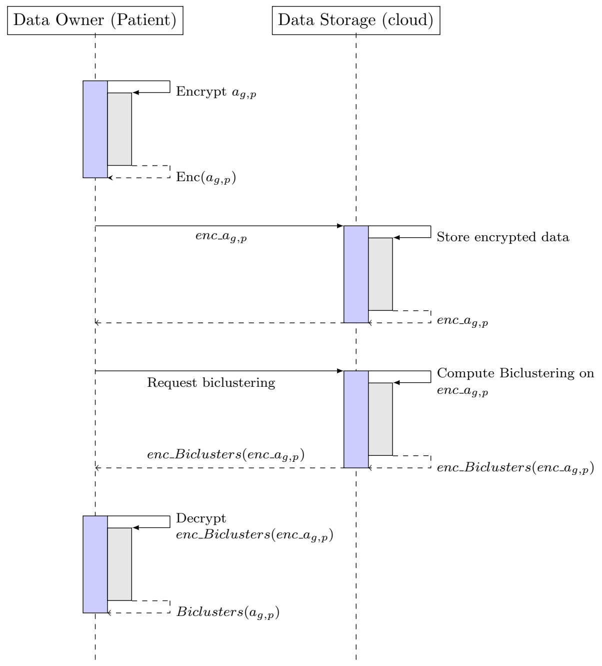


Fig. 1: Message Flow for Secure Processing of CCA

Our Secured CCA (SeCCA) proceeds along the same steps as the original non-secured biclustering algorithm [2]; however, in our approach, any computation – in particular, the mean squared residue score – is performed over encrypted data. It takes an input matrix (encrypted homomorphically on the cell level $a_{g,p}$), number of biclusters, MSR threshold (maximum mean squared residue accepted (δ parameter in the original study)), scaling factor, and minimum number of columns. It returns the list of biclusters for the given data set.

4.2 Implementation

We here represent our proposed secure analysis of Cheng and Church biclustering algorithm based on homomorphic encryption as the main contribution

Table 2: SeCCA Steps

Step	Main Idea / Description
1	HE over maximum mean squared residue accepted $(((enc\_maxValue - enc\_minValue)^2)/12) \cdot 0.005$
2	Computation of mean squared residue score of rows, columns and full data matrix: $\frac{1}{\lvert I\rvert\lvert J\rvert}\sum_{i\in I, j\in J}((enc\_a_{ij}) - (enc\_a_{iJ}) - (enc\_a_{Ij}) + (enc\_a_{IJ}))^2$
3	Computation of mean squared residue score of columns for node addition step $\frac{1}{\lvert I\rvert}\sum_{i\in I}((enc\_a_{ij}) - (enc\_a_{iJ}) - (enc\_a_{Ij}) + (enc\_a_{IJ}))^2$
4	Computation of mean squared residue score of the rows and the inverse of the rows for node addition step: $\frac{1}{\lvert J\rvert}\sum_{j\in J}((enc\_a_{ij}) - (enc\_a_{iJ}) - (enc\_a_{Ij}) + (enc\_a_{IJ}))^2$ and $\frac{1}{\lvert J\rvert}\sum_{j\in J}(-(enc\_a_{ij}) + (enc\_a_{iJ}) - (enc\_a_{Ij}) + (enc\_a_{IJ}))^2$
5	HE over rows to remove in multiple node deletion step $\frac{1}{\lvert J\rvert}\sum_{j\in J}((enc\_a_{ij}) - (enc\_a_{iJ}) - (enc\_a_{Ij}) + (enc\_a_{IJ}))^2 > \alpha(enc\_H(I, J))$
6	HE over columns to remove in multiple node deletion step $\frac{1}{\lvert I\rvert}\sum_{i\in I}((enc\_a_{ij}) - (enc\_a_{iJ}) - (enc\_a_{Ij}) + (enc\_a_{IJ}))^2 > \alpha(enc\_H(I, J))$
7	HE over rows to add in node addition step $\frac{1}{\lvert J\rvert}\sum_{j\in J}((enc\_a_{ij}) - (enc\_a_{iJ}) - (enc\_a_{Ij}) + (enc\_a_{IJ}))^2 \leq (enc\_H(I, J))$
8	HE over the inverse of the rows to add in node addition step $\frac{1}{\lvert J\rvert}\sum_{j\in J}(-(enc\_a_{ij}) + (enc\_a_{iJ}) - (enc\_a_{Ij}) + (enc\_a_{IJ}))^2 \leq (enc\_H(I, J))$
9	HE over the columns to add in node addition step $\frac{1}{\lvert I\rvert}\sum_{i\in I}((enc\_a_{ij}) - (enc\_a_{iJ}) - (enc\_a_{Ij}) + (enc\_a_{IJ}))^2 \leq (enc\_H(I, J))$

to a project done by Padilha et al. [16] which provides us by the implementation of the original algorithm, yeast cell cycle data set and Clustering Error (CE). Our procedure provides an encrypted version of the algorithm in which the efficient node-deletion algorithm was introduced to search for submatrices in expression data with low mean squared residue score [2]. Based on different types of computations needed in the procedure that can potentially be transferred to the public cloud services, we summarise the required steps of SeCCA with homomorphic encryption operations in Table 2; therefore, one can choose an appropriate computation step for a desired task. Our main focus is on steps 1 to 4 to find out how to secure the calculation of mean squared residue in different stages. Although homomorphic encryption provides an elegant solution in machine learning in theory, however, several limitations hamper its implementation; a particular limitation in real-world platforms is the lack of evaluating comparisons and conditional selections [17] on encrypted data – this limitation also applies to the Pyfhel platform we used in our experiments. We propose a multi-round interactive execution of the conditional checking part by involving the data owner to decrypt a small integer and compare it to the plaintext 0. The workflow is interactive due to restrictions of the chosen framework in support of full workflow. While it will be the main goal of our future to obtain a non-interactive evaluation of the branching conditionals, we would like to emphasise that the non-interactive workflow can also cause computational overhead and be more expensive to follow compared to the interactive fashion (e.g., employing optimised CPU). Besides, division homomorphically into a plaintext in Pyfhel is possible for fractional encoding. In this study, we rewrite steps 5 to 9 having conditional statements to remove/add nodes into a non-encrypted approach (executed on the data owner side) and present schematic diagrams of these steps in Figures 2 to 4 that is similar to the workflow (depicted in Figure 1).

Step 1. In step 1, encrypted versions of minimum and maximum values of the input data matrix are utilised for further computation of the maximum mean squared residue accepted. Here in this step, the data owner sends encrypted minimum and maximum values of the data matrix (similar to message 2 in Figure 1) to the public cloud service because of the problems mentioned earlier. Then, computation of step 1 occurs in the storage system by having encrypted input parameters (i.e., enc_maxValue and enc_minValue); the result can be returned to the data owner or can be reused by the cloud storage for further computation:

$$enc\_MSR\_thr = (((enc\_maxValue - enc\_minValue)^2)/12) \cdot 0.005$$

The data owner can decrypt and obtain the result by private key:
$Dec_{sk}(enc\_MSR\_thr)$.

Step 2. To measure the mean squared residue score (step 2), first, we need to compute the mean of the rows, the columns, and the submatrix over encrypted values. The data owner defines a submatrix of the input data ($submatrix_{ij}$) and passes it after encryption within a NumPy array to the storage system; there, the mean value of the rows (axis=1 of encrypted submatrix), the columns (axis=0 of encrypted submatrix) and the submatrix are determined by homomorphic encryption over NumPy arrays. Hence, predefined arithmetic operations (e.g., $-$, $+$) to calculate residue ($enc\_residue_{ij}$) and squared residue ($enc\_squaredResidue_{ij}$) of elements in the data matrix are applied in the ciphertext. For each submatrix cell $a_{i,j}$ we hence obtain its encrypted residue as well as squared residue (confer the plaintext counterparts in Section 3.1):

$$\begin{aligned} enc\_residue_{ij} &= enc\_submatrix_{ij} - enc\_rowMeans_i - enc\_colMeans_j \\ &\quad + enc\_submatrixMean \\ enc\_squaredResidue_{ij} &= (enc\_residue_{ij})^2 \end{aligned}$$

Afterwards, the storage system continues the process of finding the mean value of the squared residue (enc_MSR) of the elements by taking the sum of the encrypted squared residue (a NumPy operation)and its length to further divide a ciphertext (sum value of the encrypted squared residue) into a plaintext (length of the encrypted squared residue) and sends back the final result to the data owner: $enc\_MSR = mean(enc\_squaredResidue_{ij})$ over all rows i and all columns j in the submatrix. Then decryption can be performed by the data owner: $Dec_{sk}(enc\_MSR)$. An approach similar to step 2 is taken for step 3 (with a difference in the computation which takes place on the subsets of the rows as well) and step 4 (computation of mean squared residue is considered for both rows and the inverse of the rows).

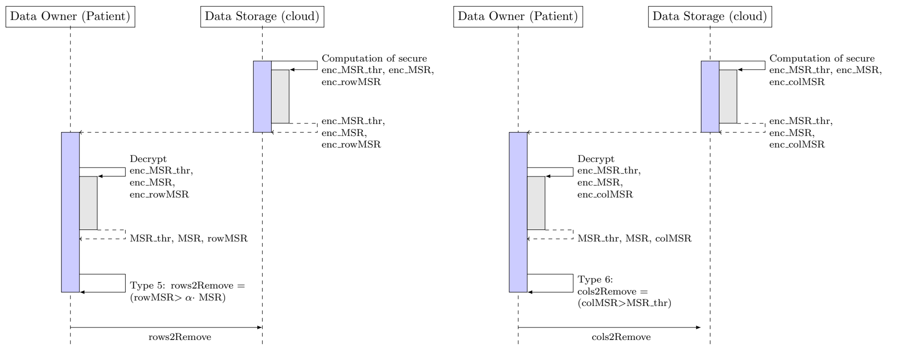

Data Owner (Patient)
Data Storage (cloud)
Computation of secure enc_MSR_thr, enc_MSR, enc_rowMSR
enc_MSR_thr, enc_MSR, enc_rowMSR
Decrypt enc_MSR_thr, enc_MSR, enc_rowMSR
MSR_thr, MSR, rowMSR
Type 5: rows2Remove = (rowMSR> α· MSR)
rows2Remove
Data Owner (Patient)
Data Storage (cloud)
Computation of secure enc_MSR_thr, enc_MSR, enc_colMSR
enc_MSR_thr, enc_MSR, enc_colMSR
Decrypt enc_MSR_thr, enc_MSR, enc_colMSR
MSR_thr, MSR, colMSR
Type 6: cols2Remove = (colMSR>MSR_thr)
cols2Remove

Fig. 2: Rows and Columns to Remove in Multiple Node Deletion Step (SeCCA steps 5 and 6)

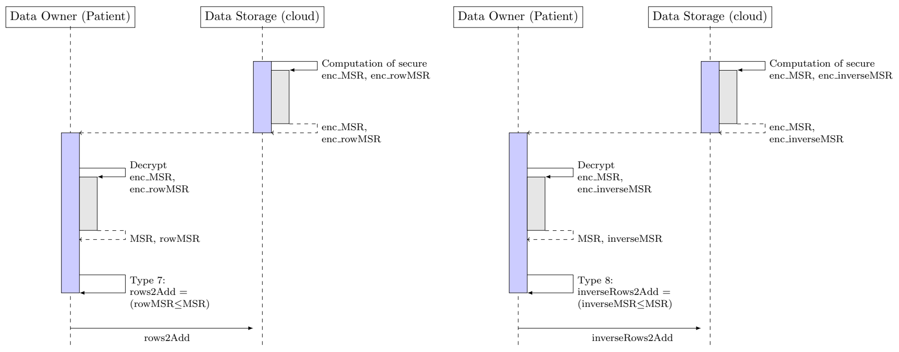

Data Owner (Patient)
Data Storage (cloud)
Computation of secure enc_MSR, enc_rowMSR
enc_MSR, enc_rowMSR
Decrypt enc_MSR, enc_rowMSR
MSR, rowMSR
Type 7: rows2Add = (rowMSR≤MSR)
rows2Add
Data Owner (Patient)
Data Storage (cloud)
Computation of secure enc_MSR, enc_inverseMSR
enc_MSR, enc_inverseMSR
Decrypt enc_MSR, enc_inverseMSR
MSR, inverseMSR
Type 8: inverseRows2Add = (inverseMSR≤MSR)
inverseRows2Add

Fig. 3: Rows and Inverted of the Rows to Add in Node Addition Step (SeCCA steps 7 and 8)

5 Experiments

In this study, we have used the yeast cell cycle expression data that was used for testing the original implementation of the algorithm [2]. The data consists of 2884 genes and 17 conditions; a matrix of integers in the range between 0 and 600 that were selected according to Tavazoie et al. [18]. Missing data are

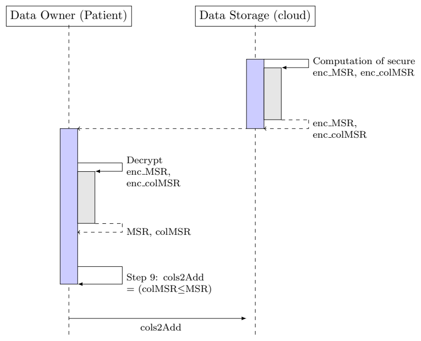
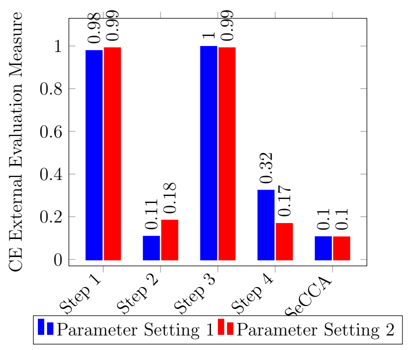

Data Owner (Patient)
Data Storage (cloud)
Computation of secure enc_MSR, enc_colMSR
enc_MSR, enc_colMSR
Decrypt enc_MSR, enc_colMSR
MSR, colMSR
Step 9: cols2Add = (colMSR≤MSR)
cols2Add

Fig. 4: Columns to Add in Node Addition Step (SeCCA step 9)

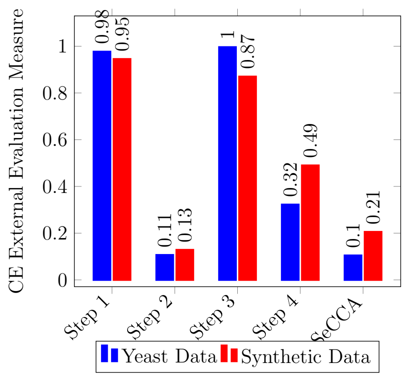
also replaced with generated random numbers that form a uniform distribution between 0 and 800. Furthermore, these random values are the candidates to get removed in the node deletion phase as they would not form recognisable patterns [2]. Additionally, we have executed our experiment on synthetic data sets based on the bicluster model (i.e., constant) according to a procedure developed by [16]. Thus, for further testing in this paper, we focus on making a constant data set consisting of 300 rows, 50 columns with 5 biclusters as a sample data set. We measured the performance of encryption and decryption processes inside SeCCA steps 1 to 4. We then carried out the comparison of the secured version of the algorithm with the original one to examine the accuracy of our proposed approaches. We used the homomorphic encryption functionalities available in Pyfhel [12] provides such as addition, multiplication, exponentiation, or scalar product that uses a similar syntax to normal arithmetic (+, -, *). The number of biclusters was determined based on the current hardware resources and without increasing the complexity of the implementation phase. Each step generates 5 biclusters, with a number of genes and conditions with a possible degree of overlap. In this paper, we apply Clustering Error (CE), achieving better results in the empirical analysis [16]. The CE measure ranges over the interval [0, 1], with higher values indicating better solutions. The CE is formulated as:

$$CE(A, \hat{A}) := \frac{d_{max}}{|U|}$$

where d_{max} is a measure of intersection of biclusterings and $|U|$ the total space covered by the biclusterings considering overlaps [3]. We perform a comparison of the four encrypted versions, steps 1 to 4 of the Cheng and Church algorithm to the non-encrypted version (as the ground truth) by CE over the input data set. By comparing step 1 of SeCCA with the non-encrypted Cheng and Church algorithm (CCA), the corresponding evaluation measure CE shows an outstanding result with a score of 0.97662. Similarly, step 3 of SeCCA represents a considerably high similarity score close to 1 with 0.99611. Despite the high similarity of steps 1 and 3 to CCA, our experiment produces striking dissimilarity among step 2 and the non-encrypted version of CCA with a score of 0.10659. We tried to improve this result by adjusting different HE parameters (like p=65537, m=4096, $base$=2, $intDigits$=16, $fracDigits$=64) as parameter setting 2 in Figure 5 and 6, resulting in a slight improvement reaching a score of 0.18207 (further information about HE parameters in Section 3.2). In Figure 5 (a), results of accuracy between two different HE parameter settings over yeast cell cycle data set are shown. An improvement can be seen in the performance of step 4 with a score of 0.32225. Consequently, the combination of the four aforementioned steps is affected by what we achieved so far in steps 2 and 4, thus showing a notable difference between SeCCA and CCA with 0.10458. Actually, the context of Pyfhel in our project is set according to available hardware resources to make the project run on a large amount of data. The settings include adapting the parameter fracDigits which specifies the number of bits that are used to encode the fractional part; thus, all the fractional parts, in case of an insufficient amount of this parameter, will be coerced into invalid result. We observed that the noise budget can reach zero soon after each squaring residue operation leading to an incorrect decrypted result. To solve these issues, we set the encryption parameters to have a high noise budget and enough fraction digits at the beginning. Still, there is room for improving accuracy in working with floating points by changing to the appropriate schemes. To compare the resulting biclusters with another data set, we choose synthetic data based on the constant bicluster model. Figure 5 (b) depicts the CE scores on yeast gene expression data and synthetic data for individually implemented steps (1 to 4) and their combinations.



(a) Different HE Parameter Settings



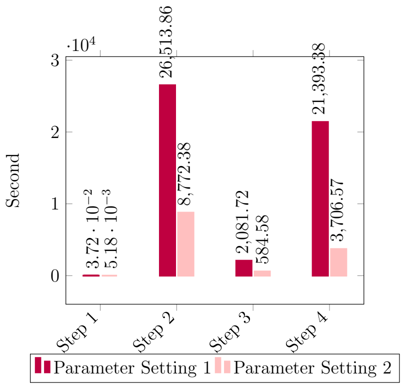
(b) Different Data sets

Fig. 5: Comparison of SeCCA with CCA by CE

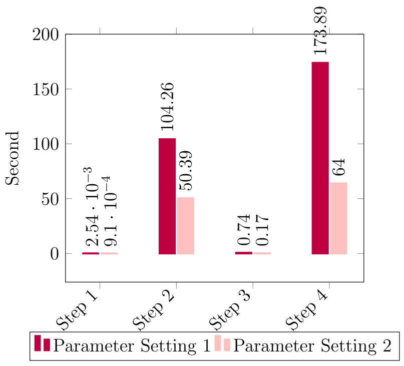
Computational Performances We conduct the experiment on a single server with Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz and 131182528 kB RAM with Rocky Linux 8.6 (Green Obsidian). The Python version used in the project is 3.8 on the PyCharm environment. Table 3 demonstrates the time performance of encryption and decryption for four implemented steps of SeCCA (steps 1 to 4) and 5 biclusters for yeast gene expression data and Table 3 for time performance of these steps on synthetic data. Hence other parts of the code in each step that run without homomorphic encryption can be found by subtracting the total execution time from the summation of encryption and decryption time. Large-scale gene expression data sets such as our two-dimensional input matrices need Pyfhel's vector operations; thus, it leads to dramatically increased execution time in performing computationally intensive tasks, including multiplication, compared to the non-encrypted Cheng and Church algorithm. As shown in Table 3, there is relatively no delay in doing encryption on maximum mean squared residue accepted homomorphically (step 1) with 0.0372 seconds in encrypting and 0.00254 for decryption of required parameters. Although, it takes a considerable amount of time for the remaining three steps to complete their tasks (encryption, decryption), where the computation of mean squared residue for node deletion/addition relies on a number of loops and different homomorphic encryption operations. Step 2 of SeCCA is one of the most time-consuming tasks by far, with 26513.85849 seconds executing encryption tasks (which also takes 132034560 in Bytes memory after creating an instance of Pyfhel) due to its applications in node deletion phase and recomputation of mean squared residue score after each update in node addition step. As steps 3 and 4 are exclusively designed for encrypting score in node addition, they show better time performance compared to step 2 by 2081.72169 and 21393.38351 seconds, respectively, although in node addition phase, inverted of the rows extends the computation to form mirror images of the rest of the rows in the bicluster results (see Section 4.2) in an increased encryption/decryption time for step 4. As discussed earlier, we carried out our experiment on different HE parameters which leads to increased accuracy of some steps. Figure 6 is provided in order to present the effects of these changes in time performance (encryption and decryption) over yeast cell cycle expression data. We also tested the execution time of individual steps (1 to 4) for the generated synthetic data that consists of 300 rows and 50 columns; Table 3 represents the encryption as well as decryption time.

Table 3: Time performance (encryption and decryption) of steps 1 to 4 of SeCCA for Yeast Gene Expression Data (left) and Synthetic Data (right)

Step	Encr. Time (Sec.)	Decr. Time (Sec.)	Step	Encr. Time (Sec.)	Decr. Time (Sec.)
1	0.0372	0.00254	1	0.03656	0.00247
2	26513.85849	104.25779	2	6918.90463	33.05609
3	2081.72169	0.73747	3	522.23543	0.85587
4	21393.38351	173.88712	4	1629.95298	12.71682



(a) Encryption Time



(b) Decryption Time

Fig. 6: Time Performance According to Different HE Parameter Settings on Yeast Cell Cycle Data

6 Conclusion and Future Work

In this paper, we proposed the Secured Cheng and Church Algorithm (SeCCA) to find a given number of biclusters with predefined parameters and showed the applicability of homomorphic encryption over biclustering algorithms – particularly Cheng and Church algorithm – with a number of steps encrypted with homomorphic operations. To the best of our knowledge, the Secured version of the Cheng and Church algorithm (SeCCA) is the first application of homomorphic encryption in biclustering algorithms that enhances the overall genomic privacy of data owners. Based on what we have achieved so far, homomorphic encryption operations are capable of calculating the maximum mean squared residue accepted and mean squared residues score in both phases (node deletion and addition) for an input data matrix with predefined parameters. Our experiment throughout steps 1 to 4 reveals a meaningful analysis of the sample yeast gene expression data and synthetic data based on a constant bicluster model with 5 biclusters and subsets of rows and columns (genes and conditions, respectively). Each step provides partial security for the entire procedure. SeCCA follows the original algorithm, but the number of biclusters is reduced to run the project in a reasonable amount of time. Moreover, nine SeCCA steps were determined to represent the overall required computations that are part of the Cheng and Church algorithm, and the storage system (public cloud service) is responsible for their execution. Among them, only four steps have specific computations that were feasible with the applied homomorphic encryption library; for the rest of the SeCCA steps, we have to rewrite the expressions in future work. Computational performance is greatly affected by the algorithm's complexity, which consists of heavy-duty computations (e.g., multiplication), leaving room for future improvements. We also compare SeCCA with the original algorithm by the external evaluation measure, Clustering Error (CE). As future work, we plan to improve the overall accuracy of SeCCA and increase the scalability of our approach (in particular regarding the number of biclusters) for which performance optimisation is required. To generate a fully secure version of the Cheng and Church algorithm, it is of great importance to come up with further practical approaches to modify statements in steps 5 to 9 straightforwardly. We will also address the computational performance in future work. One possible solution to optimise performance is the packing mechanism. SIMD has been implemented for integers in Pyfhel==2.3.1; however, homomorphic encryption operations rely on floating-point numbers in our application so that, we will consider the CKKS scheme with SIMD packing for improved performance in terms of computational overhead and accuracy. Moreover, we aim to generalise privacy-preserving gene expression data analysis by extending our approach to other biclustering algorithms and developing a secure biclustering platform with the aim to achieve a profound impact on personalised medicine, regardless of existing limitations of current homomorphic encryption libraries.

References

1. Jose-Garcia, Adan and Jacques, Julie and Sobanski, Vincent and Dhaenens, Clarisse: Biclustering Algorithms Based on Metaheuristics: A Review. arXiv preprint arXiv:2203.16241 (2022)
2. Cheng, Yizong and Church, George M.: Biclustering of expression data. Ismb **8**(2000) pp. 93–103, (2000)
3. Nicholls, Kath and Wallace, Chris: Comparison of sparse biclustering algorithms for gene expression datasets. Briefings in bioinformatics **22**(6) pp. bbab140 (2021)
4. Naveed, Muhammad and Ayday, Erman and Clayton, Ellen W and Fellay, Jacques and Gunter, Carl A and Hubaux, Jean-Pierre and Malin, Bradley A and Wang, XiaoFeng: Privacy in the genomic era. ACM Computing Surveys (CSUR) **48**(1) pp. 1–44 (2015)
5. Tu, Wangshu and Subedi, Sanjeena: A family of mixture models for biclustering. Statistical Analysis and Data Mining: The ASA Data Science Journal **15**(2) pp. 206–224 (2022)
6. Maâtouk, Ons and Ayadi, Wassim and Bouziri, Hend and Duval, Béatrice: Evolutionary biclustering algorithms: an experimental study on microarray data. Soft Computing **23**(17) pp. 7671–7697 (2019)
7. Ngo, Michelle N and Pluta, Dustin S and Ngo, Alexander N and Shahbaba, Babake: Conjoined Dirichlet Process. arXiv preprint arXiv:2002.03223 (2020)
8. Byun, J and Lee, J and Park, S: Privacy-preserving evaluation for support vector clustering. Electronics Letters **57**(2) pp. 61–64 (2021)
9. Lee, Joon-Woo and Kang, HyungChul and Lee, Yongwoo and Choi, Woosuk and Eom, Jieun and Deryabin, Maxim and Lee, Eunsang and Lee, Junghyun and Yoo, Donghoon and Kim, Young-Sik and others: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. IEEE Access **10** pp. 30039–30054 (2022)
10. Tu, Zheng and Wang, Xu An and Su, Yunxuan and Li, Ying and Liu, Jiasen: Toward Secure K-means Clustering Based on Homomorphic Encryption in Cloud. International Conference on Emerging Internetworking, Data & Web Technologies pp. 52–62 (2022)

11. Zhang, Chengliang and Li, Suyi and Xia, Junzhe and Wang, Wei and Yan, Feng and Liu, Yang: BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning. 2020 USENIX annual technical conference (USENIX ATC 20) pp. 493–506 (2020)
12. Ibarrondo, Alberto and Viand, Alexander: Pyfhel: Python for homomorphic encryption libraries. Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography pp. 11–16 (2021)
13. Brakerski, Zvika and Gentry, Craig and Vaikuntanathan, Vinod: (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3) pp. 1–36 (2014)
14. Paverd, Andrew and Martin, Andrew and Brown, Ian: Modelling and automatically analysing privacy properties for honest-but-curious adversaries. Technical Report (2014)
15. Demirci, Huseyin and Lenzini, Gabriele: Privacy-preserving Copy Number Variation Analysis with Homomorphic Encryption. (2022)
16. Padilha, Victor A and Campello, Ricardo JGB: A systematic comparative evaluation of biclustering techniques. BMC bioinformatics **18**(1) pp. 1–25 (2017)
17. Chialva, Diego and Dooms, Ann: Conditionals in homomorphic encryption and machine learning applications. arXiv preprint arXiv:1810.12380 (2018)
18. Tavazoie, Saeed and Hughes, Jason D and Campbell, Michael J and Cho, Raymond J and Church, George M.: Systematic determination of genetic network architecture. Nature genetics **22**(3) pp. 281–285 (1999)
19. Bozdemir, B and Canard, S and Ermis, O and Möllering, H and Önen, M and Schneider, Th.: Privacy-preserving density-based clustering. Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security 658–671 (2021)
20. Mohassel, P and Rosulek, M and Trieu, N.: Practical privacy-preserving k-means clustering. Cryptology ePrint Archive (2019)
21. Jäschke, A and Armknecht, F.: Unsupervised machine learning on encrypted data. International Conference on Selected Areas in Cryptography 453–478 (2018)
22. Perscheid, C and Uflacker, M.: Integrating biological context into the analysis of gene expression data. International Symposium on Distributed Computing and Artificial Intelligence 339–343 (2018)