Arithmetic PCA for Encrypted Data

Jung Hee Cheon[1,3], Hyeongmin Choe[1], Saebyul Jung[1], Duhyeong Kim*[2], Dah Hoon Lee[1], and Jai Hyun Park[1]

[1] Seoul National University, Seoul, Republic of Korea
{jhcheon, sixtail528, newstar0329, dahoon46, jhyunp}@snu.ac.kr
[2] Intel Labs, Hillsboro, OR, United States
duhyeong.kim@intel.com
[3] CryptoLab Inc.

Abstract. Reducing the size of large dimensional data is a critical task in machine learning (ML) that often involves using principal component analysis (PCA). In privacy-preserving ML, data confidentiality is of utmost importance, and reducing data size is a crucial way to cut overall costs.
This work focuses on minimizing the number of normalization processes in the PCA algorithm, which is a costly procedure in encrypted PCA. By modifying Krasulina's algorithm, non-polynomial operations were eliminated, except for a single delayed normalization at the end.
Our PCA algorithm demonstrated similar performance to conventional PCA algorithms in face recognition applications. We also implemented it using the CKKS (Cheon-Kim-Kim-Song) homomorphic encryption scheme and obtained the first 6 principal components of a 128×128 real matrix in 7.85 minutes using 8 threads.

1 Introduction

Principal component analysis (PCA) is one of the most popular tools in machine learning (ML) for dimensionality reduction. One can regard PCA as an orthogonal linear transformation that converts the coordinate system of the data set to the new coordinate system. The direction corresponds to the greatest variance via projection to the direction on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on [16]. Note that the components are related to the top k eigenvectors of the covariance matrix. One can compress the data using the first several principal components without losing the principal information.

PCA is applied in various fields, from neuroscience to quantitative finance, and the most common application is facial recognition. As with other machine learning techniques, personal private data may concern some personal information leakage. Especially for neuroscience or face recognition, since the personal information in the data cannot be discarded or changed, it is essential to consider data protection.

* This work was done while the author was affiliated with Seoul National University.

Homomorphic encryption (HE) is an encryption scheme that supports computations of encrypted messages without decrypting them. Due to the property, HE is considered a valuable tool for privacy-preserving ML [1, 3, 5, 8, 14, 15, 17] in the computation delegation scenarios in the client-server model. When applying HE to ML, the input data size should be reduced as in the plain case. Hence, evaluating PCA over HE can be an essential problem for ML using large-scale data.

However, most HE libraries [7, 10, 12, 24] mainly support arithmetic operations only, such as additions and multiplications. Hence, linear and division operations necessary for PCA cannot be directly applied. One can evaluate linear operations efficiently using a particular packing method [15]. However, if the matrix-matrix or matrix-vector multiplications were iteratively applied and if the size of the matrix is large, it might be hard to keep the packing structure. One can evaluate the division operation via a polynomial approximation of the division function that minimizes the error in a particular interval. However, this can be more problematic since several divisions are necessary for most PCA algorithms.

1.1 Our contribution.

Arithmetic PCA algorithm. In this work, we modified a PCA algorithm called Krasulina's method, reducing the number of divisions and maintaining only one delayed division. We give the correctness proof of our arithmetic algorithm that converges to an eigenvector corresponding to the maximal eigenvalue. The delayed division in our algorithm can even be treated as post-processing in the k-PCA scenario, which finds the k eigenvectors correspond to the k maximum eigenvalues. With no divisions during the computation, the client can obtain the eigenvectors by normalizing them after decryption.

We applied our PCA algorithm to face recognition using the Yale face data [13]. We compare the face recognition accuracy based on our PCA algorithm with other PCA algorithms, such as Krasulina's and the power methods. Compared to the state-of-the-art PCA algorithms, ours achieved almost the same R2 score and accuracy for face recognition.

HE evaluation of k-PCA for large matrix. We implemented our PCA algorithm for top k principal components on encrypted data with the CKKS homomorphic encryption scheme. Our algorithm can deal with much larger data than the prior works using power methods using maximum 20×20 matrices; we implemented it for 128×128 size positive semi-definite matrix. We get the first 6 principal components within 12.1 minutes, with numerically accurate eigenvectors satisfying $\|A\mathbf{v} - \lambda\mathbf{v}\|_\infty < 0.012$. The run-time of our implementation is at least 1.2 times faster than the estimated time of Panda's implementation [21] on the Yale face data[4].

[4] Note that their run-time is 19.9 minutes, but the 168 bootstrappings are excluded, and the estimated run-time of their implementation is at least 9.2 minutes, including the bootstrappings. However, it is more likely to be close to 13 minutes since they use non-bootstrapable parameters.

1.2 Prior works.

Homomorphic PCA based on power method. Pereira and Aranha [22] proposed an algorithm for performing PCA on encrypted data. Compared to the original power method updating the vector $\mathbf{v}$ as $\mathbf{v} \leftarrow A\mathbf{v}/\|\mathbf{v}\|$, they modified it by reducing the normalizations and divide with predetermined constants as $\mathbf{v} \leftarrow A\mathbf{v}/c_i$ for i th iteration. However, finding the proper constants c is another eigenvalue problem, ironically, since if they are not close to the eigenvalues λ, the size of $\mathbf{v}$ will diverge or converge to 0. They also modified the algorithm for updating A to find the next principal component by reducing the normalization.

Several works [20, 23] improved the power method approach. Notably, Rathee et al. [23] very efficiently implemented it with HElib, resulting in 6.5 seconds with 8 threads but only for a very small matrix of size 20×20. Moreover, since the size of the eigenvector increases exponentially on the number of iterations (a.k.a. epoch), they could only perform a maximum of 5 iterations and only for the first principal component.

Panda [21] and Lu et al. [19] also implemented the HE-friendly PCA algorithm based on the power method; however, they need extra communications between the client and the server. In a general client-server scenario for computation delegation, the client encrypts the data and sends it to the server once. Then, the server computes homomorphically and sends it back to the client. However, Panda and Lu et al. make the server send an intermediate ciphertext to the client, and the client decrypt-then-encrypt it and sends it back. This was to bypass the expensive bootstrapping procedure; however, security concerns may occur due to the increased communications, and the communication cost becomes huge due to the size of the ciphertexts. Panda's implementation with the CKKS scheme achieved 6 eigenvectors with an R2 score of 0.579 within 19.9 minutes, but 168 bootstrappings were replaced with decrypt-then-encrypt. The total time, including the bootstrapping, is estimated as at least 13.59 minutes with 8 threads. We note that this is a rough lower bound driven by adding the bootstrapping time. Since their HE parameters were not bootstrapable, the actual run-time of all the operations will likely blow up significantly.

2 Preliminaries

Notations. Define the polynomial rings $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ for positive integer N and q, where N is a power of 2. Also, we define an isomorphism $\Phi : \mathbb{R}[X]/(X^N+1) \to \mathbb{C}^{N/2}$ which maps $m(X) \mapsto \left(m(\zeta^{5^j})\right)_{0 \le j < N/2}$, where $m \in \mathbb{R}[X]$ and $\zeta = e^{\frac{\pi i}{2N}}$ is a primitive $2N$-th root of unity. All the vectors are column vectors and are denoted as bold lowercase alphabets unless they are the columns of some predetermined matrix.

2.1 Homomorphic encryption

Homomorphic encryption (HE) is an encryption scheme that allows algebraic calculations on encrypted ciphertexts. Homomorphic evaluation of an operation on the ciphertexts produces a new ciphertext, which encrypts the result plaintexts of the operation. That is, for ciphertexts $\mathsf{ct}_1, \cdots, \mathsf{ct}_n \in \mathcal{C}$ encrypting the messages $m_1, \cdots, m_n \in \mathcal{M}$ and a circuit $C : \mathcal{M}^n \to \mathcal{M}$, the following relation holds:

$$\mathsf{Dec}(\mathsf{Eval}(\mathcal{C}, \mathsf{ct}_1, \cdots, \mathsf{ct}_n)) = C(m_1, \cdots, m_n).$$

In general, the ciphertext has an essential noisy term (error) that grows according to the depth of the circuit $\mathcal{C}$. If the magnitude of the error becomes bigger than some threshold, the homomorphic property is not guaranteed. However, by using a special operation called bootstrapping, the size of the error can be decreased. Thus, bootstrapable HE can support arbitrary computations, and we call it Fully Homomorphic Encryption (FHE). Craig Gentry first proposed the concrete construction of HE, and after that, various HE schemes and libraries were proposed [4, 7, 9, 11].

CKKS scheme. CKKS scheme is an approximate HE scheme that can handle real-valued data [7, 6]. Since all the current exact HE schemes can only deal with finite field elements, the approximate HE scheme is used popularly for practical computations, including machine learning. The simplified schematic diagram of the CKKS scheme is given as follows:

$$\begin{array}{ccccccc} \mathcal{M} & & \underset{\mathsf{Decode}}{\overset{\mathsf{Encode}}{\rightleftharpoons}} & & \mathcal{P} & \underset{\mathsf{Dec}}{\overset{\mathsf{Enc}}{\rightleftharpoons}} & \mathcal{C} \\ \mathbb{C}^{\frac{N}{2}} (\text{or } \mathbb{R}^N) & \overset{\Delta, \Phi}{\longleftrightarrow} & \mathbb{R}[X]/(X^N+1) & \overset{\lfloor \cdot \rceil}{\longleftrightarrow} & \mathbb{Z}[X]/(X^N+1) & \leftrightarrow & \mathcal{R}_q^2 \end{array}$$

The data $\mathbf{z}$ is in $\mathbb{C}^{N/2}$ (or can be viewed as $\mathbb{R}^N$) and $\|\mathbf{z}\|_\infty \leq 1$. Then $\mathbf{z}$ can be scaled by a scaling factor Δ, and let m be the corresponding polynomial in $\mathbb{R}[X]/(X^N+1)$ by the isomorphism Φ. Then, this can be rounded to a plaintext $m \in \mathbb{Z}[X]/(X^N+1)$. We call this mapping and its approximate inverse as an encoding (Encode) and decoding (Decode) and define as follows:

$\mathsf{Encode}(\mathbf{z}; \Delta)$: For $\mathbf{z} \in \mathbb{C}^{N/2}$, compute $\mathsf{pt} = \lfloor \Phi^{-1}(\Delta \cdot \mathbf{z}) \rceil_{\mathcal{R}} \in \mathcal{R}$.
$\mathsf{Decode}(\mathsf{pt}; \Delta)$: For $\mathsf{pt} \in \mathcal{R}$, compute $\mathbf{z} = \Delta^{-1} \cdot \Phi(\mathsf{pt}) \in \mathbb{C}^{N/2}$.

Then, the encryption can be applied to the plaintext. Algorithms in the CKKS scheme are described as follows:

$\mathsf{KeyGen}(1^\lambda)$: Sample s uniformly from $\{0, \pm 1\}^N$ with Hamming weight h (i.e., $\|s\|_1 = h$), and set secret key $\mathsf{sk} \leftarrow (1, s) \in \mathcal{R}^2$. Sample u, u' and u_i's uniformly from $\mathcal{R}_q$ and e, e' and e_i's from an error distribution χ and set public key $\mathsf{pk} \leftarrow (-u \cdot s + e, u) \in \mathcal{R}_q^2$. For an integer P, set evaluation key $\mathsf{evk} \leftarrow (-u' \cdot s + e' + Ps^2, u') \in \mathcal{R}_{Pq}^2$ and rotation keys $\mathsf{rotk}_i \leftarrow (-u_i \cdot s + e_i + Ps^{5^i}, u_i) \in \mathcal{R}_{Pq}^2$.

$\mathsf{Enc}(\mathsf{pt};\mathsf{pk})$: First sample $v \leftarrow \mathcal{R}_q$ uniformly and compute $\mathsf{ct} = v \cdot \mathsf{pk} + (\mathsf{pt} + e_0,\ e_1) \in \mathbb{R}_q^2$, where the error e_0 and e_1 are sampled from χ.

$\mathsf{Dec}(\mathsf{ct};\mathsf{sk})$: For $\mathsf{ct} = (c_0, c_1)$ and $\mathsf{sk} = (1, s)$, compute $\mathsf{pt} = c_0 + c_1 \cdot s$.

$\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2)$: For two ciphertexts ct_1, ct_2, compute $\mathsf{ct}_{\mathsf{add}} = \mathsf{ct}_1 + \mathsf{ct}_2 \mod q$.

$\mathsf{Mult}(\mathsf{ct}_1, \mathsf{ct}_2;\mathsf{evk})$: For two ciphertexts $\mathsf{ct}_1 = (c_0, c_1)$ and $\mathsf{ct}_2 = (c_0', c_1')$, compute $\mathsf{ct}_{\mathsf{mult}} = (c_0 \cdot c_0',\ c_0' \cdot c_1 + c_0 \cdot c_1') + \lfloor \frac{c_1 \cdot c_1' \cdot \mathsf{evk}}{P} \rceil \in \mathcal{R}_q^2$.

$\mathsf{Rot}_i(\mathsf{ct};\mathsf{rotk}_i)$: For $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_q^2$, compute $\mathsf{ct}_{\mathsf{rot},i} = (c_0, 0) + \lfloor \frac{c_1 \cdot \mathsf{rotk_i}}{P} \rceil \in \mathcal{R}_q^2$.

Note that the corresponding message vector $\mathbf{z} = (z_1, z_2, \cdots, z_{N/2})$ will be permuted into $\mathbf{z}_{\mathsf{rot},i} = (z_{i+1}, z_{i+2}, \cdots, z_{N/2}, z_1, \cdots, z_i)$ via Rot_i operation.

2.2 Principal component analysis

Principal component analysis (PCA) is one of the methods of statistical techniques of dimensionality reduction and feature extraction, which converts the high-dimensional space of the data to a low-dimensional one while preserving the distribution of existing data, i.e., the variance as much as possible. PCA converts the previous variables into new variables by using a linear combination.

Suppose the data to be analyzed has d variables and N samples. We can present the data as a matrix A, where $A \in \mathbb{R}^{d \times N}$. Now, let $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_d \in \mathbb{R}^{d \times 1}$ be coefficient vectors, and $P = (\mathbf{p}_1 | \mathbf{p}_2 | \cdots | \mathbf{p}_d)$ be a $d \times d$ matrix. Suppose the result of the projection of A on P is represented as follows:

$$Z = P^t A = \begin{bmatrix} \mathbf{p}_1{}^t \\ \mathbf{p}_2{}^t \\ \vdots \\ \mathbf{p}_d{}^t \end{bmatrix} A.$$

Since we need to preserve the variance as much as possible, we need to maximize the variance of each row of the matrix Z, i.e.,

$$\mathbf{z}_1 = \mathbf{p}_1{}^t A,\ \mathbf{z}_2 = \mathbf{p}_2{}^t A,\ \cdots,\ \mathbf{z}_d = \mathbf{p}_d{}^t A.$$

Let Σ be the covariance matrix of the data matrix A. Then we need to find the coefficient vectors $\mathbf{p}_i$ for $i = 1, 2, \cdots, d$ which satisfy the following:

$$\max_{\mathbf{p}_i} \mathrm{Var}(\mathbf{z}_i) = \max_{\mathbf{p}_i} \mathbf{p}_i{}^t \Sigma \mathbf{p}_i, \quad \|\mathbf{p}_i\| = \mathbf{p}_i{}^t \mathbf{p}_i = 1.$$

To solve the equation above, let $L_i = \mathbf{p}_i{}^t \Sigma \mathbf{p}_i - \lambda(\mathbf{p}_i{}^t \mathbf{p}_i - 1)$. Then the $\mathbf{p}_i$ maximizing $\mathrm{Var}(\mathbf{z}_i)$ satisfies

$$\frac{\partial L_i}{\partial \mathbf{p}_i} = \Sigma \mathbf{p}_i - \lambda \mathbf{p}_i = (\Sigma - \lambda)\mathbf{p}_i = 0.$$

By the definition of the eigenvalue and the eigenvector, $\mathbf{p}_i$ should be the eigenvector of Σ. We call the eigenvector of the Σ principal component, and by using these principal components in each column of matrix P, we can get the maximal variance. It is well known that every covariance matrix is a non-singular matrix, and the number of eigenvectors exists as the number of dimensions d, so the value of $\mathbf{p}_i$'s are valid. Then we extract k-maximal eigenvalues ($k \leq d$) and the eigenvectors correspond to each eigenvalue.

Krasulina's algorithm. Krasulina's algorithm is one of the methods for stochastic approximation, which is an iterative process to find the minimum eigenvalue of a symmetrical matrix [18]. For a symmetric matrix $A \in \mathbb{R}^{m\times m}$, let the eigenvalues as $\lambda_1 < \cdots < \lambda_s$ and the corresponding eigenspaces as E_{λ_i} for $1 \leq i \leq s$. For an initial random vector $\mathbf{v}_0 \in \mathbb{R}^m$, Krasulina's algorithm updates the vector as follows:

$$\mathbf{v}_{n+1} = \mathbf{v}_n - \frac{\gamma_n}{m}\left\{A - \frac{(A\mathbf{v}_n, \mathbf{v}_n)}{\|\mathbf{v}_n\|^2} I\right\}\mathbf{v}_n,$$

where $\gamma_n > 0$ is a sequence of constants. It is known that $\mathbf{v}_n$ converges to an eigenvector in E_{λ_1}, if the following conditions hold:

i) $\|\pi_1(\mathbf{v}_0)\| \neq 0$ via the projection map $\pi_1 : \mathbb{R}^d \to E_{\lambda_1}$,
ii) $\sum_{n=1}^{\infty} \gamma_n = \infty$ and $\sum_{n=1}^{\infty} \gamma_n^2 < \infty$.

Face recognition. Face recognition is a supported application program to identify and classify each person with a digital image. It is done by comparing the database of faces and the selected features of the person. [26]

Suppose the image file of the face has $k \times l$ size. The images are saved as a flattened vector, and each image pixel has a particular numerical value. Suppose the image is now a $d \times 1$ vector, say $\mathbf{x}_i$ $(i = 1, 2, \cdots, N)$, where $d = k \cdot l$ and N is the number of samples. Then we concatenate the vectors as $X = (\mathbf{x}_1|\mathbf{x}_2|\cdots|\mathbf{x}_N)$ where $X \in \mathbb{R}^{d\times N}$. We first center the samples by computing $X \leftarrow X - (\bar{\mathbf{x}} \mid \bar{\mathbf{x}} \mid \cdots \mid \bar{\mathbf{x}})$, where $\bar{\mathbf{x}} \in \mathbb{R}^d$ is the mean vector of the vector $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$, i.e. $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$. Then, we can write the covariance matrix of X as XX^t. For the eigenvalues $\lambda_1 < \lambda_2 < \cdots < \lambda_s$ of the covariance matrix XX^t, we find k-maximal eigenvalues and eigenvectors that correspond to the eigenvalues. It is well known that when $d \gg N$, we can use X^tX rather than XX^t to get a meaningful eigenvectors, since $XX^t(X\mathbf{v}_i) = X(X^tX\mathbf{v}_i) = \lambda_i(X\mathbf{v}_i)$, when letting $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_N$ be the eigenvectors of X^tX [25, 26].

Following the analysis above, one can reconstruct the eigenfaces $\mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_k \in \mathbb{R}^d$ such that

$$\mathbf{f}_i = \sum_{j=1}^{N} \mathbf{v}_{ij}\mathbf{x}_j, \quad i = 1, 2, \cdots, k,$$

where $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k$ are the eigenvectors of the matrix X^tX which correspond to k-maximal eigenvalues.

Then, we can use the eigenfaces to classify a new face image. Suppose a new face image $\mathbf{y} \in \mathbb{R}^d$ is given. Then, for $i = 1, 2, \cdots, k$ $(k \leq N)$, its i-th eigenface component is represented as $w_i = \mathbf{f}_i{}^t (\mathbf{y} - \bar{\mathbf{x}})$. Now, we obtain an eigenface representation $\mathbf{w}(\mathbf{y}) \coloneqq (w_1, w_2, \cdots, w_k)^t$ of the input face $\mathbf{y}$. Let $\mathbf{w}_i = \mathbf{w}(\mathbf{x}_i)$ for $i = 1, 2, \cdots, N$ [26].

Now, one can classify the face image $\mathbf{y}$ to one of the N faces by finding the index i that minimizes the distance between the eigenface representations $\mathbf{w}$ and $\mathbf{w}_i$ as

$$\mathsf{idx} = \underset{1 \leq i \leq N}{\operatorname{argmin}} \|\mathbf{w} - \mathbf{w}_i\|.$$

3 Arithmetic PCA algorithm

Original PCA algorithms require many divisions or normalizations during the iterations. However, in terms of homomorphic evaluation, a division is expensive compared to additions and multiplications. Thus, reducing the number of divisions is crucial for the algorithm's efficiency. We, therefore, modify Krasulina's algorithm to a HE-friendly algorithm by reducing the number of divisions. As a result, it only requires one delayed division at the end. In this section, we first introduce the modified Krasulina's algorithm, which we call the *Division-free Krasulina algorithm*. Since the convergence of the iterative PCA algorithms significantly depends on the size of the vectors, we provide a convergence proof of our algorithm.

3.1 Division-free Krasulina algorithm

We here introduce the modified division-free variant of Krasulina's algorithm, which has no normalization in each iteration process. We change the updating equation of each iteration by multiplying ξ_n by $\|\mathbf{v}_n\|^2$, which is shown as follows:

$$\mathbf{v}_{n+1} = \mathbf{v}_n - \frac{\gamma_n}{N}\{\|\mathbf{v}_n\|^2 A - (A\mathbf{v}_n, \mathbf{v}_n)I\}\mathbf{v}_n, \tag{3.1}$$

or equivalently,

$$\xi_n = \|\mathbf{v}_n\|^2 A\mathbf{v}_n - (A\mathbf{v}_n, \mathbf{v}_n)\mathbf{v}_n, \tag{3.2}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n - \frac{\gamma_n}{N}\xi_n. \tag{3.3}$$

The original Krasulina's algorithm provides a method for determining the minimum eigenvalue of a symmetric matrix. However, PCA requires finding k-maximal eigenvalues of the covariance matrix. So, we first modify the algorithm to find an eigenvector that corresponds to the largest eigenvalue as

$$\xi_n = \|\mathbf{v}_n\|^2 A\mathbf{v}_n - (A\mathbf{v}_n, \mathbf{v}_n)\mathbf{v}_n, \tag{3.4}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \frac{\gamma_n}{N}\xi_n. \tag{3.5}$$

Then, to find the eigenvectors that correspond to the top k eigenvalues, we update the covariance matrix after finding each eigenvector $\mathbf{v}$, as

$$A \leftarrow \|\mathbf{v}\|^2 A - A\mathbf{v}\mathbf{v}^t. \tag{3.6}$$

Note that the update of A is just orthogonalizing against the eigenvector $\mathbf{v}$ to eliminate the $\mathbf{v}$-direction component. The algorithm can be applied to the covariance matrix XX^t or X^tX to find its top k eigenvectors.

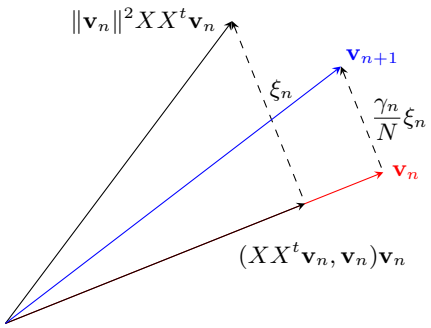

Fig. 1: Update policy of the modified method

3.2 Convergence proof

It is well known that any positive semi-definite matrix A can be represented as $A = XX^t$, so we consider the matrix XX^t in the convergence of the Division-free Krasulina algorithm. For $X \in \mathbb{R}^{d\times N}$, we define $\mu(V) := \frac{\mathbf{v}^tA\mathbf{v}}{\|\mathbf{v}\|^2} = \frac{\|X^t\mathbf{v}\|^2}{\|\mathbf{v}\|^2}$ for column vector $\mathbf{v} \in \mathbb{R}^d$. Also, let $\lambda_1 < \lambda_2 < \cdots < \lambda_s$ be the distinct eigenvalues of XX^t. Now, we state the main theorem and a corollary.

Theorem 1. *Let $A \in \mathbb{R}^{d\times d}$ be the positive semi-definite matrix and $\lambda_1 < \lambda_2 < \cdots < \lambda_s$ be the distinct eigenvalues of A. Assume $\dim(E_{\lambda_s}) = 1$, $\mathbf{v}_0 \notin \cup_{i\neq s} E_{\lambda_i}$ and $\gamma_n > 0$ is $\Theta(1/n)$. Then $\mathbf{v}_n \to \mathbf{u}$ for some $\mathbf{u} \in E_{\lambda_s}$ as $n \to \infty$.*

Corollary 1. *Under the same condition of Theorem 1, $\mu(\mathbf{v}_n) \to \lambda_s$ as $n \to \infty$, where $\mu(\mathbf{v}) := \dfrac{\mathbf{v}^tA\mathbf{v}}{\|\mathbf{v}\|^2}$.*

Proof. This is proven immediately by Theorem 1, since

$$\mu(\mathbf{v}_n) = \frac{\|X^t\mathbf{v}_n\|^2}{\|\mathbf{v}_n\|^2} \to \frac{\|X^t\mathbf{u}\|^2}{\|\mathbf{u}\|^2} = \frac{(XX^t\mathbf{u}, \mathbf{u})}{\|\mathbf{u}\|^2} = \lambda_s$$

for some $\mathbf{u} \in E_{\lambda_s}$ as $n \to \infty$. □

We first see some properties we need before starting the proof of Theorem 1.

Lemma 1. *For every $n \in \mathbb{N}$, ξ_n is orthogonal to $\mathbf{v}_n$ and $\|\mathbf{v}_{n+1}\| \geq \|\mathbf{v}_n\|$ holds.*

Proof. Since

$$\begin{aligned}\xi_n &= \|\mathbf{v}_n\|^2 XX^t\mathbf{v}_n - (XX^t\mathbf{v}_n, \mathbf{v}_n)\mathbf{v}_n \\ &= \|\mathbf{v}_n\|^2 \left(XX^t\mathbf{v}_n - \frac{(XX^t\mathbf{v}_n, \mathbf{v}_n)}{(\mathbf{v}_n, \mathbf{v}_n)}\mathbf{v}_n\right)\end{aligned}$$

holds for every $n \in \mathbb{N}$, $(\xi_n, \mathbf{v}_n) = 0$ holds. Hence ξ_n is orthogonal to $\mathbf{v}_n$ and thus by Pythagoras theorem, $\|\mathbf{v}_{n+1}\| \geq \|\mathbf{v}_n\|$ holds. □

Lemma 2. *If $\gamma_n > 0$ is $\mathcal{O}(1/n)$, then $\|\mathbf{v}_n\| < c \cdot n^{1/5}$ for every sufficiently large n where $c > 0$ is a constant.*

Proof. From figure 1, we can deduce

$$\|\mathbf{v}_{n+1}\|^2 = \|\mathbf{v}_n\|^2 + \frac{\gamma_n^2}{N^2}\|\xi_n\|^2, \tag{3.7}$$

$$\|\xi_n\|^2 = \|\mathbf{v}_n\|^4 \cdot \|XX^t\mathbf{v}_n\|^2 - \|X^t\mathbf{v}_n\|^4 \cdot \|\mathbf{v}_n\|^2, \tag{3.8}$$

and since $\|XX^t\mathbf{v}_n\| \leq \|XX^t\|_{\sup} \cdot \|\mathbf{v}_n\|$, we get

$$\begin{aligned}\|\mathbf{v}_{n+1}\|^2 &\leq \|\mathbf{v}_n\|^2 \left(1 + \frac{\gamma_n^2}{N^2}\|\mathbf{v}_n\|^2 \cdot \|XX^t\mathbf{v}_n\|^2\right) \\ &\leq \|\mathbf{v}_n\|^2 \left(1 + \frac{\gamma_n^2}{N^2}\|XX^t\|_{\sup}^2 \cdot \|\mathbf{v}_n\|^4\right) \\ &= \|\mathbf{v}_n\|^2 \left(1 + c_0\gamma_n^2\|\mathbf{v}_n\|^4\right),\end{aligned}$$

where $c_0 = \|\frac{1}{N}XX^t\|_{\sup}^2$ be a constant. Since $\gamma_n = \mathcal{O}(1/n)$, there is a constant $c_1 > 0$ such that $\gamma_n < \frac{c_1}{n}$ for every sufficiently large n. If we assume $\|\mathbf{v}_n\| < c \cdot n^{1/5}$ for sufficiently large n and $c > 0$,

$$\begin{aligned}\|\mathbf{v}_{n+1}\|^2 &\leq \|\mathbf{v}_n\|^2 \left(1 + c_0\gamma_n^2\|\mathbf{v}_n\|^4\right) \\ &< c^2 n^{2/5} \left(1 + c_0 \cdot \frac{c_1^2}{n^2} \cdot c^4 \cdot n^{4/5}\right) \\ &= c^2(n+1)^{2/5} \cdot \left(\frac{n}{n+1}\right)^{2/5} \cdot \left(1 + c_0c_1^2c^4 \cdot n^{-6/5}\right).\end{aligned}$$

Let $h(n) = \frac{2}{5}n^{1/5} - \frac{3}{25}n^{-4/5}$, then $h(n)$ strictly increases for positive n, and diverges to ∞ when $n \to \infty$. Thus for sufficiently large n, $h(n) > c_0c_1^2c^4$ and thus

$$1 + c_0c_1^2c^4 \cdot n^{-6/5} < 1 + \frac{2}{5}n^{-1} - \frac{3}{25}n^{-2} < \left(1 + \frac{1}{n}\right)^{2/5},$$

which implies $\|\mathbf{v}_{n+1}\|^2 < c \cdot (n+1)^{2/5}$. Hence, inductively, the lemma holds. □

Lemma 3. *Under the same conditions as in Lemma 2,* $\|\mathbf{v}_n\|$ *converges to a positive real value.*

Proof. Recall that $\|\mathbf{v}_{n+1}\|^2 \leq \|\mathbf{v}_n\|^2 \left(1 + c_0\gamma_n^2\|\mathbf{v}_n\|^4\right)$. By Lemma 2, for a sufficiently large n,

$$\|\mathbf{v}_{n+1}\|^2 \leq \|\mathbf{v}_n\|^2 \left(1 + c_2 \cdot n^{-6/5}\right),$$

for some $c_2 > 0$. Note that $\prod_n \left(1 + c_2 \cdot n^{-6/5}\right)$ converges since

$$\begin{aligned}\log \prod_n \left(1 + c_2 \cdot n^{-6/5}\right) &= \sum_n \log \left(1 + c_2 \cdot n^{-6/5}\right) \\ &< c_2 \sum_n n^{-6/5} < \infty,\end{aligned}$$

and hence $\|\mathbf{v}_n\|$ is bounded. From Lemma 1, $\|\mathbf{v}_n\|$ increases and thus converges to a positive real value. □

Lemma 4. *Under the same conditions as in Lemma 2,* $\alpha_n \coloneqq (\mathbf{u}, \mathbf{v}_n)$ *converges for every eigenvector* $\mathbf{u} \in E_{\lambda_s}$.

Proof. Let $\mathbf{u} \in E_{\lambda_s}$ be an eigenvector corresponds to λ_s and assume $\alpha_1 \geq 0$. By the definition, $XX^t\mathbf{u} = \lambda_s\mathbf{u}$ holds, and we have

$$\begin{aligned}\alpha_{n+1} &= (\mathbf{u}, \mathbf{v}_n + \frac{\gamma_n}{N}\xi_n) = \alpha_n + \frac{\gamma_n}{N}(\mathbf{u}, \xi_n) \\ &= \alpha_n + \frac{\gamma_n}{N}\left(\mathbf{u},\ \|\mathbf{v}_n\|^2 XX^t\mathbf{v}_n - \|X^t\mathbf{v}_n\|^2\mathbf{v}_n\right) \\ &= \alpha_n + \frac{\gamma_n}{N}\left\{\|\mathbf{v}_n\|^2(\lambda_s\mathbf{u}, \mathbf{v}_n) - \|X^t\mathbf{v}_n\|^2\alpha_n\right\} \\ &= \alpha_n \left\{1 + \frac{\gamma_n}{N}\left(\lambda_s\|\mathbf{v}_n\|^2 - \|X^t\mathbf{v}_n\|^2\right)\right\}. \end{aligned} \tag{3.9}$$

Since λ_s is the largest eigenvalue of XX^t, it holds that $\|X^t\mathbf{v}_n\|^2 \leq \lambda_s\|\mathbf{v}_n\|^2$ and thus α_n increases. Moreover, $\alpha_n = (\mathbf{u}, \mathbf{v}_n) \leq \|\mathbf{u}\| \cdot \|\mathbf{v}_n\|$ is bounded, and thus α_n converges. □

We now prove Theorem 1 using the abovementioned lemmas.

Proof *(Proof of Theorem 1).* Let $\mathbf{u} \in E_{\lambda_s}$ be an eigenvector corresponds to λ_s. By the result of Lemma 4, $\alpha_n = (\mathbf{u}, \mathbf{v}_n)$ converges. Without loss of generality, we assume that $\alpha_1 \geq 0$. Since $(\mathbf{u}, \mathbf{v}_n) = \|\mathbf{u}\| \cdot \|\mathbf{v}_n\| \cdot \cos\theta_n$ and $\|\mathbf{v}_n\|$ both converge, $\cos\theta_n$ also converges.

Suppose $\lim_{n\to\infty} \cos\theta_n < 1$. Then there exists $M_0 > 0$ such that if $n > M_0$ then there exist a constant $k_n > 0$ and a vector $\mathbf{w}_n \in \mathbb{R}^d$ which satisfy the following:

$$\mathbf{w}_n \perp \mathbf{u}, \quad \mathbf{v}_n = k_n\mathbf{u} + \mathbf{w}_n, \quad \|\mathbf{w}_n\| \neq 0, \quad \mathbf{w}_n \notin E_{\lambda_s}. \tag{3.10}$$

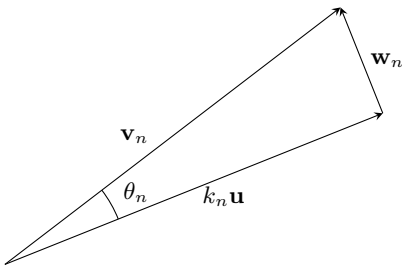

Fig. 2: Classification of the vector $\mathbf{w}_n$

Note that $k_n \to k$ for some $k > 0$ as $n \to \infty$, since

$$\alpha_n = (\mathbf{u}, \mathbf{v}_n) = (\mathbf{u}, k_n\mathbf{u} + \mathbf{w}_n) = k_n\|\mathbf{u}\|^2$$

converges. Then, we have

$$\begin{aligned}\mu(\mathbf{v}_n) &= \frac{\|X^t\mathbf{v}_n\|^2}{\|\mathbf{v}_n\|^2} \\ &= \frac{\|k_nX^t\mathbf{u} + X^t\mathbf{w}_n\|^2}{\|k_n\mathbf{u} + \mathbf{w}_n\|^2} \\ &= \frac{k_n^2\|X^t\mathbf{u}\|^2 + \|X^t\mathbf{w}_n\|^2}{k_n^2\|\mathbf{u}\|^2 + \|\mathbf{w}_n\|^2},\end{aligned} \tag{3.11}$$

since $(X^t\mathbf{u}, X^t\mathbf{w}_n) = (XX^t\mathbf{u}, \mathbf{w}_n) = \lambda_s(\mathbf{u}, \mathbf{w}_n) = 0$. Also, we obtain that

$$\begin{aligned}\mu(\mathbf{v}_n) &= \frac{1}{k_n^2 + \frac{\|\mathbf{w}_n\|^2}{\|\mathbf{u}\|^2}} \cdot \left(k_n^2\lambda_s + \frac{\|\mathbf{w}_n\|^2}{\|\mathbf{u}\|^2} \cdot \frac{\|X^t\mathbf{w}_n\|^2}{\|\mathbf{w}_n\|^2}\right) \\ &< \frac{1}{k_n^2 + \frac{\|\mathbf{w}_n\|^2}{\|\mathbf{u}\|^2}} \cdot \lambda_s\left(k_n^2 + \frac{\|\mathbf{w}_n\|^2}{\|\mathbf{u}\|^2}\right) = \lambda_s,\end{aligned} \tag{3.12}$$

since $\dim E_{\lambda_s} = 1$ and $\mathbf{w}_n \notin E_{\lambda_s}$. So, there exist $\epsilon_0 > 0$ such that if $n > M_0$ then $\mu(\mathbf{v}_n) < \lambda_s - \epsilon_0$. From (3.9), we have

$$\begin{aligned}\alpha_{n+1} &= \alpha_n\left\{1 + \frac{\gamma_n}{N}\left(\lambda_s\|\mathbf{v}_n\|^2 - \|X^t\mathbf{v}_n\|^2\right)\right\} \\ &= \alpha_n\left\{1 + \frac{\gamma_n}{N}\|\mathbf{v}_n\|^2\left(\lambda_s - \mu(\mathbf{v}_n)\right)\right\} > \alpha_n\left\{1 + \frac{\epsilon_0 \cdot \gamma_n}{N}\|\mathbf{v}_n\|^2\right\},\end{aligned}$$

and hence for any $m > n > M_0$,

$$\alpha_m - \alpha_n > \sum_{i=n}^{m-1} \frac{\epsilon_0 \cdot \gamma_i}{N}\|\mathbf{v}_i\|^2\alpha_i.$$

From Lemma 3, $\lim_{n\to\infty} \|\mathbf{v}_n\| > 0$ implies that there exist $\epsilon_1 > 0$ and $M_1 > 0$ such that if $n > M_1$ then $\|\mathbf{v}_n\| > \epsilon_1$. Therefore for any $m > n > \max(M_0, M_1)$,

$$\begin{aligned}\alpha_m - \alpha_n &> \sum_{i=n}^{m-1} \frac{\epsilon_0 \cdot \gamma_i}{N} \|\mathbf{v}_i\|^2 \alpha_i \\ &> \frac{\epsilon_0 \epsilon_1^2 \alpha_n}{N} \sum_{i=n}^{m-1} \gamma_i,\end{aligned}$$

and thus $\alpha_m \to \infty$ as $m \to \infty$ since $\gamma_n = \Theta(1/n)$, which is a contradiction. Hence $\lim_{n\to\infty} \cos\theta_n = 1$ and $\lim_{n\to\infty} \mathbf{w}_n = \mathbf{0} \in \mathbb{R}^d$. Thus we get

$$\lim_{n\to\infty} (\mathbf{v}_n) = \lim_{n\to\infty} (k_n \mathbf{u} + \mathbf{w}_n) = k\mathbf{u}.$$

Therefore we conclude that $\lim_{n\to\infty} \mathbf{v}_n = \mathbf{u}$ for some $\mathbf{u} \in E_{\lambda_s}$. □

4 Implementation

In this section, we give the specific algorithms for HE implementation of the division-free Krasulina algorithm and the results. We adopted the algorithms to use SIMD operation on the HE scheme for efficiency. We first introduce some basic algorithms for SIMD computation. Note that our experiment is two-folded: i) we first implement PCA over encrypted data in Section 5.1, and ii) run the face recognition part in plaintext in Section 5.2.

4.1 HE implementation

All the algorithms deal with length N vectors corresponding to N slots in each ciphertext. The vectors $v = (v_0, \cdots, v_{m-1}) \in \mathbb{R}^m$ are replicated N/m times in the N slots as $(v\|v\|\cdots\|v) = (v_0, \cdots, v_{m-1}, v_0, \cdots, v_{m-1}, \cdots\cdots, v_0, \cdots, v_{m-1})$, where m and N are power of 2. We first give some basic algorithms RotSum1, 2 and Replicate in Algorithms 1, 2 and 3.

Algorithm 1: Rotate and Sum 1 (RotSum1)

Input : a vector $v = (v_0, \cdots, v_{N-1}) \in \mathbb{R}^N$ and a power-of-two integer $m|N$.
Output : a vector $(w\|w\|\cdots\|w) \in \mathbb{R}^N$ for $w \in \mathbb{R}^m$ such that $w[i] = \sum_{j=0}^{N/m-1} v_{i+mj}$ for all $i = 0, \cdots, m-1$.

$w \leftarrow v$
for $0 \le i < \log N - \log m$ **do**
  $w \leftarrow \mathsf{Add}(w, \mathsf{Rot}_{m\times 2^i}(w))$
end
return w

Algorithm 2: Rotate and Sum 2 (RotSum2)

Input : a vector $v = (v_0, \cdots, v_{N-1}) \in \mathbb{R}^N$ and a power-of-two integer $m|N$.
Output : a vector $(w_0, w_1 \cdots w_{N-1}) \in \mathbb{R}^N$ such that $w_i = \sum_{j=0}^{m-1} v_{(i+j)\%N}$ for all $i = 0, \cdots, m-1$.

$w \leftarrow v$
for $0 \leq i < \log m$ **do**
  $w \leftarrow \mathsf{Add}(w, \mathsf{Rot}_{2^i}(w))$
end
return w

Algorithm 3: Replicate (Replicate)

Input : a vector $v' = (v \| v \| \cdots \| v) \in \mathbb{R}^N$ for $v = (v_0, \cdots, v_{m-1}) \in \mathbb{R}^m$.
Output : a vector $w = (v_0, \cdots, v_0, v_1, \cdots, v_1, \cdots\cdots, v_{m-1}, \cdots, v_{m-1}) \in \mathbb{R}^N$.

$z \leftarrow v'$
$w \leftarrow 0$
for $0 \leq i < m$ **do**
  $z \leftarrow \mathsf{Rot}_{N-1}(z)$
  $w \leftarrow \mathsf{Add}(w, \vec{e}_{N-im-1} \cdot z)$ // $\vec{e}_t[i] = 1$ if $i = t$, 0 otherwise
end
$w \leftarrow \mathsf{RotSum2}(w, m)$
return w

Algorithm 4: Inner Product (InnProd)

Input : two vectors $v, w \in \mathbb{R}^n$.
Output : $z = (a, a, \cdots, a) \in \mathbb{R}^n$ where $a = v \cdot w$.

$z \leftarrow \mathsf{Mult}(v, w)$
$z \leftarrow \mathsf{RotSum2}(z, n)$
return z

For computing the norm of the vector, or the inner product $\mathbf{v}^t(A\mathbf{v})$, we here give the algorithm for the inner product in Algorithm 4.

For the linear operations in the updating equation, we give two algorithms matrix-vector multiplication (MatVecMult) for $A\mathbf{v}$ and vector-vector multiplication (VecVecTrsMult) for $(A\mathbf{v})\mathbf{v}^t$. It is important to maintain the structure of the matrix A when updating it, and need to use the corresponding matrix-vector multiplication algorithm.

Using the above building blocks, we now give a concrete k-PCA algorithm using the division-free Krasulina algorithm in Algorithm 7.

4.2 Privacy-preserving face recognition scenario

In our scenario for face recognition, two main parties are involved - a client and a server, as in general computation delegation scenarios. The client has some

Algorithm 5: Matrix-Vector Multiplication (MatVecMult)

Input : a vector $c = (c_1 \| \cdots \| c_m) \in \mathbb{R}^N$ for column vectors $c_i \in \mathbb{R}^m$ of a matrix $A \in \mathbb{R}^{m \times m}$ and a vector $v' = (v \| \cdots \| v) \in \mathbb{R}^N$, where $N = m^2$.

Output : a vector $(Av \| \cdots \| Av) \in \mathbb{R}^N$, where $Av \in \mathbb{R}^m$.

$z \leftarrow \mathsf{Mult}(c, \mathsf{Replicate}(v'))$
$z \leftarrow \mathsf{RotSum1}(z, m)$
return z

Algorithm 6: Vector-Vector Multiplication (VecVecTrsMult)

Input : two vectors $v' = (v \| \cdots \| v), w' = (w \| \cdots \| w) \in \mathbb{R}^N$ for $v, w \in \mathbb{R}^m$, where $m^2 = N$.

Output : a vector $c = (c_1 \| \cdots \| c_m) \in \mathbb{R}^N$ for column vectors $c_i \in \mathbb{R}^m$ of a matrix $vw^t \in \mathbb{R}^{m \times m}$.

$v' \leftarrow \mathsf{Mult}(\mathsf{Replicate}(v'), w')$
return v'

Algorithm 7: Division-free Krasulina algorithm for top k eigenvalues (kPCA)

Input : the integer $k > 0$, a vector $c = (c_1 \| \cdots \| c_m) \in \mathbb{R}^N$ for column vectors $c_i \in \mathbb{R}^m$ of a positive semi-definite matrix $A \in \mathbb{R}^{m \times m}$ and learning rates $\{\gamma_{i,j}\}_{1 \le i \le k}$.

Output : vectors $(v_i \| \cdots \| v_i) \in \mathbb{R}^N$ for the eigenvectors v_i's correspond to the k maximal eigenvalues of A.

for $1 \le i \le k$ **do**
  $v_i \leftarrow (v_i \| \cdots \| v_i)$ for $v_i \xleftarrow{\$} [0,1]^m$ `// sample a random initial vector`
  for $1 \le j \le \mathsf{epoch}_i$ **do** `// for some estimated` epoch_i
    $\mathsf{tmp}_1 \leftarrow \mathsf{InnProd}(v_i, v_i)$
    $w \leftarrow \mathsf{MatVecMult}(c, v_i)$
    $\mathsf{tmp}_2 \leftarrow \mathsf{InnProd}(v_i, w)$
    $\xi \leftarrow \mathsf{Mult}(\mathsf{tmp}_1, w) - \mathsf{Mult}(\mathsf{tmp}_2, v_i)$
    $v_i \leftarrow v_i + \gamma_{i,j} \cdot \xi$
  end
  if $i \neq k$ **then**
    $w \leftarrow \mathsf{MatVecMult}(c, v_i)$
    $c' \leftarrow \mathsf{VecVecTrsMult}(w, v_i)$
    $\mathsf{tmp}_1 \leftarrow \mathsf{InnProd}(v_i, v_i)$
    $c \leftarrow \mathsf{Mult}(\mathsf{tmp}_1, c) - c'$ `// update` A `by orthogonalizing against` v_i
  end
end
return $\{v_i\}_{1 \le i \le k}$

face image data from a group of people, and for a given new face image, it wants

to classify the new image as one of the people in the group. The client can then use PCA to recognize the faces, while its computational power is somewhat limited. Thus, the client wants to delegate PCA to the server with much stronger computational power.

The problem is that the client needs to give all the private face data to the server and may leak personal information. So, the client uses HE to encrypt the face images and then sends the ciphertexts to the server. Then, the server will homomorphically evaluate PCA on the encrypted data. The client will receive the ciphertext of the eigenvectors and decrypt it. Now, the client can classify the new image with just a few computations. The scenario is given below. Note that the indices i and j are defined for $1 \leq i \leq k$ and $1 \leq j \leq N$.

Training step.

Client: For given face data $X \in \mathbb{R}^{d \times N}$ which is a concatenation of the N samples $\mathbf{x}_i \in \mathbb{R}^d$, it computes
1) $X \leftarrow X - (\bar{\mathbf{x}}|\cdots|\bar{\mathbf{x}})$ where $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$,
2) $A \leftarrow X^t X \in \mathbb{R}^{N \times N}$,
3) $\mathsf{sk}, \mathsf{pk}, \mathsf{evk}, \mathsf{rotk}_i\mathsf{s} \leftarrow \mathsf{KeyGen}$,
4) $\mathsf{ct} \leftarrow \mathsf{Enc}(A; \mathsf{pk})$,

and sends ct with $\mathsf{pk}, \mathsf{evk}, \mathsf{rotk}$ to the Server. // ct includes training data

Server: For given ct and $\mathsf{pk}, \mathsf{evk}, \mathsf{rotk}_i$'s, it computes $\mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{kPCA}, \mathsf{ct}; \mathsf{evk}, \mathsf{rotk}_i\text{'s})$ and sends ct' back to the client. // ct' includes non-normalized eigenvectors

Client: For given ct, it computes
1) $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k \leftarrow \mathsf{Dec}(\mathsf{ct}', \mathsf{sk})$,
2) $\mathbf{v}_i \leftarrow \mathbf{v}_i / \|\mathbf{v}_i\|$,
3) $\mathbf{f}_i \leftarrow \sum_{j=1}^{N} \mathbf{v}_{ij}\mathbf{x}_j$,
4) $w_{ji} \leftarrow \mathbf{f}_i^t(\mathbf{x}_j - \bar{\mathbf{x}})$,
5) $\mathbf{w}_j \leftarrow (w_{j1}, \cdots, w_{jk})^t$.

Inference step. // $\mathbf{y}$ maybe encrypted w.r.t. owner's HE key

Client: When a new face image $\mathbf{y} \in \mathbb{R}^d$ is given, it computes
1) $w_i \leftarrow {\mathbf{f}_i}^t(\mathbf{y} - \bar{\mathbf{x}})$,
2) $\mathbf{w} \leftarrow (w_1, \cdots, w_k)^t$,
3) find $\mathsf{idx} \leftarrow \underset{1 \leq j \leq N}{\operatorname{argmin}} \|\mathbf{w} - \mathbf{w}_j\|$,

and classify $\mathbf{y}$ to one of the N samples via index idx.

Fig. 3: Privacy-preserving face recognition scenario.

We note that the inference step of the face recognition can also be done in a privacy=preserving manner by using HE. Each face image can be encrypted using the image owner's homomorphic encryption key, and the index can be computed in an encrypted state.

5 Experiment

In this section, we give two different experiments and their results. In Section 5.1, we implement our PCA algorithm, the division-free Krasulina over encrypted data. Then, in Section 5.2, we run the face recognition using the division-free Krasulina algorithm in plaintext to guarantee the correctness and convergence of our new k-PCA algorithm.

5.1 PCA over encrypted data

In this section, we implement the modified Krasulina algorithm (Algorithm 3) using the CKKS scheme. We first perform the division-free Krasulina algorithm on 128×128 symmetric matrices with significant principal components.

We generate symmetric matrices with significant principal components for this experiment and perform the division-free Krasulina algorithm. The symmetric matrices with significant principal components are generated as follows. We sample a random real orthonormal matrix Q, i.e., $QQ^T = Q^TQ = I$, and a diagonal matrix D such that

$$D[i,j] = \begin{cases} 0 & \text{if } i \neq j \\ \lambda_i & \text{otherwise} \end{cases},$$

where $\lambda_1 > \lambda_2 > \cdots > \lambda_6 > 0$ are relatively big while $\lambda_i = 0.01$ for $i > 6$. Our sample matrix, $A = QDQ^T$, is a symmetric matrix with eigenvalues λ_i's.

To check the validity of our eigenvector v, we compute the error term as:

$$\text{error} = \|A\tilde{v} - (\tilde{v}^t A\tilde{v})\tilde{v}\|_\infty$$

where $\tilde{v}$ is the normalized vector of v, i.e., $\tilde{v} = v/\|v\|_2$. Once v is an eigenvalue of A, $\tilde{v}^t A\tilde{v}$ will be the corresponding eigenvalue, and the error term will vanish. Conversely, if the error term is sufficiently small, v is almost an eigen vector of A since $Av \approx \lambda v$ where $\lambda = \tilde{v}^t A\tilde{v}$.

Parameter Selection. We found the 6 largest principal components of given 128×128 matrix A using the division-free Krasulina algorithm (Algorithm 7). We used the learning rate of 0.05. We conducted two experiments with different numbers of iterations. The first experiment iterates 40 to find the largest principal component and 20 for the other 5 principal components. The second experiment iterates 15 iterations for all 6 principal components.

For the CKKS implementation, we used the quotient polynomial ring $\mathbb{Z}_Q[X]/(X^N + 1)$ with the dimension $N = 2^{15}$. The size of maximum modulus, $\log(PQ)$ is 777. We sampled the ternary secret key with a hamming weight of 192. This parameter is 128-bit secure [2]. We note that the CKKS scheme with the selected parameter has 9 precision bits.

All experiments were performed on an Intel Xeon Silver 4114 CPU at 2.20GHz processor.

Result. Our division-free Krasnulina algorithm found all 6 principal components of the given encrypted 128×128 matrix, A. The experimental results are reported in Table 1 and 2. Table 1 shows the results with more iterations, while Table 2 shows the results with fewer iterations. More iterations provide a better performance, but the result with fewer iterations also shows a reasonable performance with a faster running time. To be more precise, with more iterations, it takes 726.8 seconds (12.1 minutes) to find the 6 principal components with 8 threads and 52 minutes with a single thread. On the other hand, the experiment with fewer iterations takes 471.62 seconds (7.85 minutes) with 8 threads, and it shows a reasonable performance. Also, we point out that the precision bits of our HE parameter are 9.

We stress that almost all previous works considered matrices with small sizes, e.g., less than 20×20. Compared to previous works, thanks to our modification, we could successfully extract the principal components of a large-size encrypted matrix (i.e., of size 128×128).

Only one previous work by Panda [21] has implemented the PCA algorithm for large matrix using HE. However, they make the server send an intermediate ciphertext to the client, and the client *decrypt then re-encrypt* it and sends it back. This is to bypass the expensive bootstrapping procedure; however, security concerns may occur due to the increased communications, and the communication cost becomes huge due to the size of the ciphertexts. Panda's implementation with the CKKS scheme achieved 6 eigenvectors with an R2 score of 0.579 within 19.9 minutes, excluding 168 bootstrappings. Including bootstrapping, the most expensive procedure for HE implementation, we expect their total run-time to be at least 9.2 minutes, but more likely to close near 12 minutes, regarding the imperfect use of 8 threads.

Table 1: PCA over encrypted data with a significant number of iterations (1 thread).

Principal component	Eigen value (exact)	Eigen value (computed)	Error	Time (sec)
1	15	15.000	0.002	856.2
2	10	10.000	0.001	442.0
3	5	4.998	0.008	442.7
4	4	4.002	0.012	442.8
5	3	3.000	0.004	440.0
6	2	2.000	0.006	409.7

Table 2: PCA over encrypted data with a practical number of iterations (8 threads).

Principal component	Eigen value (exact)	Eigen value (computed)	Error	Time (sec)
1	15	14.992	0.044	84.25
2	10	10.019	0.067	78.91
3	5	4.961	0.041	78.92
4	4	4.059	0.047	78.91
5	3	2.998	0.010	78.57
6	2	2.005	0.021	72.06

5.2 Face Recognition.

The experiment is done with 128 plaintext samples, and a similar number of samples are distributed for each person. For $k = 4, 6, 8, 10$, Table 3 shows the k-PCA result.

Table 3: Face recognition accuracy for Yale Face (in plaintext)

Methods	score	Ours R2 / acc	Krasulina R2 / acc	Power R2 / acc
k	4	0.451 ±0.009 / 0.719	0.451 ±0.004 / 0.681	0.452 ±0.003 / 0.692
	6	0.581 ±0.010 / 0.870	0.589 ±0.001 / 0.868	0.588 ±0.002 / 0.868
	8	0.655 ±0.008 / 0.881	0.666 ±0.002 / 0.889	0.668 ±0.003 / 0.892
	10	0.703 ±0.011 / 0.892	0.716 ±0.001 / 0.892	0.718 ±0.001 / 0.892

We show the accuracy of our method (division-free Krasulina algorithm) and compare it with other methods (original Krasulina's algorithm and power method) with limited epochs - 40 for the first component and 20 for the others. *R2* indicates the R2 score with variances of the eigenvectors, and *acc* indicates the accuracy of the face recognition from the following scoring method: Let the number of test sets t, and let $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_t$ be the images from the test set. Also, define $\delta(\mathbf{y})$ as

$$\delta(\mathbf{y}) \coloneqq \begin{cases} 1 & \text{if } \mathsf{idx}(\mathbf{y}) = \mathsf{idx}_{\mathsf{real}} \\ 0 & \text{if } \mathsf{idx}(\mathbf{y}) \neq \mathsf{idx}_{\mathsf{real}} \end{cases}$$

where $\mathsf{idx}_{\mathsf{real}}$ denotes the actual index in the sample. We calculate the accuracy in Table 3 as $\frac{1}{t}\sum_{i=1}^{t} \delta(\mathbf{y}_i)$.

References

1. Al Aziz, M.M., Alhadidi, D., Mohammed, N.: Secure approximation of edit distance on genomic data. BMC medical genomics **10**(2), 55–67 (2017)
2. Albrecht, M.R., Curtis, B.R., Deo, A., Davidson, A., Player, R., Postlethwaite, E.W., Virdia, F., Wunderer, T.: Estimate all the {LWE, NTRU} schemes! In: International Conference on Security and Cryptography for Networks. pp. 351–367. Springer (2018)
3. Bonte, C., Vercauteren, F.: Privacy-preserving logistic regression training. BMC medical genomics **11**(4), 13–21 (2018)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
5. Chen, H., Gilad-Bachrach, R., Han, K., Huang, Z., Jalali, A., Laine, K., Lauter, K.: Logistic regression over encrypted data from fully homomorphic encryption. BMC medical genomics **11**(4), 3–12 (2018)
6. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 360–384. Springer (2018)
7. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International conference on the theory and application of cryptology and information security. pp. 409–437. Springer (2017)
8. Cheon, J.H., Kim, D., Park, J.H.: Towards a practical cluster analysis over encrypted data. In: Paterson, K.G., Stebila, D. (eds.) Selected Areas in Cryptography – SAC 2019. pp. 227–249. Springer International Publishing, Cham (2020)
9. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: international conference on the theory and application of cryptology and information security. pp. 3–33. Springer (2016)
10. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
11. Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: International Workshop on Public Key Cryptography. pp. 1–16. Springer (2012)
12. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Annual Cryptology Conference. pp. 75–92. Springer (2013)
13. Georghiades, A.S., Belhumeur, P.N., Kriegman, D.J.: From few to many: Generative models for recognition under variable pose and illumination. In: Proceedings fourth ieee international conference on automatic face and gesture recognition (cat. no. pr00580). pp. 277–284. IEEE (2000)
14. Hong, S., Park, J.H., Cho, W., Choe, H., Cheon, J.H.: Secure tumor classification by shallow neural network using homomorphic encryption. BMC genomics **23**(1), 1–19 (2022)
15. Jiang, X., Kim, M., Lauter, K., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. pp. 1209–1222 (2018)
16. Jolliffe, I.T.: Principal component analysis for special types of data. Springer (2002)
17. Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.H.: Logistic regression model training based on the approximate homomorphic encryption. BMC medical genomics **11**(4), 23–31 (2018)

18. Krasulina, T.: Analysis of asymptotic properties of potential function method algorithms intended to restore functional generator characteristics by points observed. Avtomat. i Telemekh (2), 114–121 (1968)
19. Liu, Y., Chen, C., Zheng, L., Wang, L., Zhou, J., Liu, G., Yang, S.: Privacy preserving pca for multiparty modeling. arXiv preprint arXiv:2002.02091 (2020)
20. Lu, W.j., Kawasaki, S., Sakuma, J.: Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. Cryptology ePrint Archive (2016)
21. Panda, S.: Principal component analysis using ckks homomorphic scheme. In: International Symposium on Cyber Security Cryptography and Machine Learning. pp. 52–70. Springer (2021)
22. Pereira, H.V., Aranha, D.F.: Principal component analysis over encrypted data using homomorphic encryption. In: Anais do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. pp. 338–341. SBC (2015)
23. Rathee, D., Mishra, P.K., Yasuda, M.: Faster pca and linear regression through hypercubes in helib. In: Proceedings of the 2018 Workshop on Privacy in the Electronic Society. pp. 42–53 (2018)
24. Microsoft SEAL (release 3.6). `https://github.com/Microsoft/SEAL` (Nov 2020), microsoft Research, Redmond, WA.
25. Sirovich, L., Kirby, M.: Low-dimensional procedure for the characterization of human faces. Josa a **4**(3), 519–524 (1987)
26. Turk, M., Pentland, A.: Eigenfaces for recognition. Journal of cognitive neuroscience **3**(1), 71–86 (1991)