Robust Matrix Completion for Privacy-Compliant Recommendation Services

Siddharth Diwan 1  Ed Bielawa 1

Abstract

In this paper, we present a privacy-compliant algorithm for robust matrix completion on non-independent-and-
identically-distributed data using a fully homomorphic encryption scheme. We provide a series of algorithms
adapting matrix completion and its robust counterpart to the fully homomorphic encryption space and integrate
these algorithms into SecRec, our previous work on privacy-compliant matrix completion for recommendation
algorithms. Evaluations of accuracy and performance indicate that our robust matrix completion algorithms
have greater recommendation accuracy when data is corrupted by an adversary in a non-independent-and-
identically-distributed fashion with a small performance cost. In the FHE setting, our robust implementation
has similar accuracy benefits, but with worse performance cost. We demonstrate a feasible prototype for robust,
privacy-preserving machine learning on the matrix completion problem. The code implementing SecRec and
its fully homomorphic encryption compatible robust matrix completion algorithms is publicly available at
https://github.com/sidwan02/SecRec .

1. Introduction

1.1. Matrix Completion

Matrix completion is a machine learning problem involving reconstruction of a low-rank matrix given access to a small
number of matrix entries and the rank of the matrix. There exist algorithms that solve the matrix completion problem for
sparse, low-rank input matrices when all revealed entries are correct (not effected by noise) and revealed in an independent
and identically distributed (IID) manner. Such algorithms adopt gradient descent methods to construct a candidate matrix \( X \)
\begin{equation}
X
\end{equation}
to minimize the candidate function:
\begin{equation}
f(X) = \|X \times^T - M\|_\Omega,
\end{equation}
where \( \|M\|_\Omega \) represents the Frobenius norm over the set of
revealed matrix indices \( \Omega \). Previous work has shown minimization of this candidate function is sufficient for an optimal
solution to the matrix completion problem (Ge et al., 2017).

One common application of matrix completion algorithms we focus on are recommendation algorithms. Recommendation
algorithms can be found in a wide variety of services, such as search engines like Google, online marketplaces like Amazon,
or movie viewing services like Netflix. Recommendation algorithms let users efficiently parse vast quantities of data by
providing them a curated list of recommendations. These services generate recommendations from previously collected user
data using algorithms like matrix completion. Completed values in this context indicate the service’s prediction of how
much a user is likely to enjoy a particular webpage, product, or movie.

1.2. Privacy Concerns

One problem with standard matrix completion algorithms is that they give such algorithms and their servers direct access to
user data. Existing recommendation algorithms collect staggering amounts of data of varying degrees of sensitivity. For
example, Google stores location data and past search history in addition to current queries (Google). Amazon collects
location data and information linked to other networked devices not directly using Amazon (Amazon.com). Netflix obtains
interaction-based data like user engagement (Netflix). Possession of such sensitive data faces problems like leakage or
sharing with advertisers.

Even if the stored user data is non-sensitive or anonymized, malicious actors can still reconstruct sensitive information about
data subjects. Researchers were able to uncover the political leanings of Netflix users despite only having access to a public
database on anonymized movie ratings (Narayanan & Shmatikov, 2006). Unfortunately, since current matrix completion
algorithms require direct access to user data, they are inseparable from these privacy concerns.
1.3. Data Assumptions

In addition to requiring insecure, plaintext versions of user data, standard matrix completion algorithms require that revealed matrix entries are revealed IID, and that these revealed entries are truthful. Such assumptions are not necessarily true in real-world data. Consider the case of movie recommendations generated from movie ratings data. Some movies are more popular than others, and therefore more likely to be rated by users. Users of a movie streaming service like Netflix watch different amounts of movies, some more than others. Movie rating data, like many other inputs to recommendation algorithm, does not follow the necessary IID assumptions required to apply standard matrix completion. We can formalize this problem by stating that some subset of our data constitutes "true” data, which is revealed in an IID manner. Then, an adversary may reveal an additional $\epsilon$ portion of our data. The adversary is not restricted to revealing data in an IID fashion.

In the movie rating example, users may also rate movies incorrectly. They may overestimate or underestimate their feelings on a given movie. We can interpret the difference between a user’s reported rating and the actual, truthful rating as noise.

While there exist variants of the matrix completion algorithm for non-IID data (the first assumption) and noisy, not truthful data (the second assumption), we primarily focus on implementing robustness for the non-IID data case. We do not address noisy data and assume that revealed data is truthful.

2. Background

2.1. Homomorphic Encryption

As previously outlined, a privacy-preserving matrix completion scheme must be able to perform computations over sensitive user data without being privy to the actual contents of said data. This problem can be solved by encrypting user data under a fully homomorphic encryption (FHE) scheme. Fully homomorphic encryption is an encryption scheme where encryption values, performing mathematical operations on them, and then decrypting the values produces an equivalent value to performing the mathematical operations, encrypting the results, then decrypting it. Under FHE, a recommendation service cannot access user data because it is encrypted, but the service can perform computations on the encrypted user data. This would allow for implementing the matrix completion algorithm in a privacy-preserving manner.

Unfortunately, there exist many limitations on algorithms one can currently implement while operating in the FHE space. Under FHE, only addition and multiplication can be performed on encrypted (ciphertext) values. Relatively elementary operations like division or square root must be approximated and have no known direct computation in FHE space. While it is theoretically possible to implement any function in FHE space by using some combination of additions and multiplications, in practice the space of algorithms one can implement in FHE space is rather limited.

Although approximations of many common functions like division or the square root exist for FHE space, these approximations are infeasible for many existing FHE libraries due to the high number of multiplications required in their implementation (Cheon et al., 2019). In our implementation of secure, robust matrix completion algorithms, we choose to use TenSEAL, a Python library for FHE derived from Microsoft’s SEAL. Like many FHE libraries, operations on ciphertexts accumulate error, or noise differentiating the decrypted ciphertext from equivalent mathematical operations performed on plaintext. We determined that one can only perform approximately five multiplications on ciphertexts in TenSEAL before this error grows larger than the true product. This limitation on multiplications further limits the amount of feasible algorithms one can implement.

FHE also has significant performance impacts on both memory and runtime compared to operations on plaintext data. Although FHE is harder to work with and has more overhead than plaintext data, it is currently the most straightforward scheme for performing privacy-preserving computations. Any implementation of robust matrix completion algorithms that also satisfy privacy compliance must therefore operate within the limitations of FHE space.

2.2. Robust Matrix Completion

Integrating FHE into matrix completion algorithms only addresses privacy concerns and not those related to user data, such as non-IID data (adversarial $\epsilon$-corruption) or noisy data. Fortunately, there exist robust matrix completion algorithms for each of these cases. These robust class of algorithms are similar to the standard matrix completion case, in that they use gradient descent to minimize a similar candidate function. Although implementations of some of these robust algorithms exist, they are designed for plaintext data rather than FHE data, and are not suited for privacy-compliant matrix completion.
Robust algorithms for matrix completion on noised data implement a post-processing step where multiple candidate completed matrices are obtained and the matrix with the smallest nuclear norm is selected. Robustness for noisy data may be particularly desirable when employing a differential privacy scheme to ensure more accurate completed matrices. We chose not to implement robust algorithms handling noisy data, though future work may want to explore implementing noise robustness in a privacy-preserving setting.

For non-IID data, we implement a robust matrix completion algorithm utilizing a pre-processing step to compute weights over revealed entries. These weights produce a spectrally similar, sparsified matrix that we then perform gradient descent on. We adapt the weight computation strategy from Ge and Cheng, but use a gradient descent method to actually compute the weights, as the weight computation method in Ge and Cheng is not feasible for implementation in FHE space (Cheng & Ge, 2018). We provide a FHE-compatible version of semi-random matrix completion using weights pre-processing.

2.3. SecRec

In our previous work, we proposed SecRec, a secure recommendation algorithm, that addresses the privacy concerns associated by matrix completion by performing the matrix completion algorithm over encrypted data using an FHE scheme. SecRec provides an interface for users to submit movie ratings and receive recommendations generated via matrix completion securely. Our work in this paper builds directly on top of SecRec, as we provide prototypes for the robust semi-random matrix completion algorithms using the encryption infrastructure provided by our SecRec prototype. We propose both FHE-compatible (for privacy) and non-FHE compatible (for performance) implementations of robust semi-random matrix completion and integrate them into SecRec.

In developing SecRec, the authors of this paper collaborated with William Sun, who implemented PIR (private information retrieval) algorithms and provided benchmarking tests. Previous work for SecRec we do not intend to present here include the server/combiner structure, as well as our initial implementation of matrix completion using FHE. Secure and insecure (using FHE and not using FHE) versions of robust semi-random matrix completion were implemented and integrated into SecRec for the sole purpose of this report rather than our initial proposal of SecRec. For more details, please see our report for CSCI 2390 - Privacy-Conscious Computer Systems, where we provide a more in-depth description and evaluation of our SecRec prototype.

3. Algorithms

We now present the algorithms used in our implementation of robust semi-random matrix completion. We provide implementations used in the FHE setting and the non-FHE setting. For algorithms used in both settings, we provide the plaintext implementation as an algorithm and discuss necessary changes to convert the plaintext algorithms to their FHE equivalent. Note that semi-random matrix completion (algorithm 1) uses weight calculation (algorithm 2) as a pre-processing step, and SVD (algorithm 3) as a subroutine. Weight calculation (2) and SVD (algorithm 3) use eigenvector extraction (algorithm 4) as a subroutine. We provide algorithms for each. Pseudocode for these algorithms can be found in section A.1.
Algorithm 1: Semi-Random Matrix Completion

**Input:** revealed recommendation matrix $M$, weight matrix $W$, features rank $r$, number of epochs $n_e$, learning rate $\alpha$.

**Notation Remark 1:** $a \leftarrow b$ means $a$ is updated to $b$

**Variables:** $M_{i,j}\in\Omega$: the set of recommendations filled by the users (the non-sparse entries).

$W_{i,j}\in\Omega$: the set of weights for individual recommendations

1) **SVD**

Compute $UDV^T = M$

Store $X = U$, $Y = V$

2) **SGD**

for $e \in 1 \ldots n_e$

for $m \in M_{i,j}\in\Omega$

Get the predicted rating $\hat{m} = \sum_{k=1}^{r} X_{i,k} Y_{j,k}$

Get the prediction error $e_{i,j} = m - \hat{m}$ and clip it appropriately.

Calculate the $X$ update $\Delta X_{i,k} = \frac{\partial e_{i,j}^2}{\partial X_{i,k}} = 2w_{i,j}e_{i,j}Y_{j,k}$.

Calculate the $Y$ update $\Delta Y_{j,k} = \frac{\partial e_{i,j}^2}{\partial Y_{j,k}} = 2w_{i,j}e_{i,j}X_{i,k}$.

With row-broadcast (omitted), update $X_{i,j} \leftarrow X_{i,j} + \alpha \Delta X_{i,k}$

With row-broadcast (omitted), update $Y_{j,k} \leftarrow Y_{j,k} + \alpha \Delta Y_{j,k}$

end for

end for

3.1. Semi-Random Matrix Completion Algorithm

In this section, we present the modified matrix completion algorithm using robust weight precomputation that we integrated into SecRec. This algorithm is derived primarily from the work of Ge and Cheng (Cheng & Ge, 2018). Note that $W$ represents a set of robust weights computed using the weight computation algorithm which we outline in the next section.

Algorithm 1 is intended to be run on plaintext inputs, and does not differ much from the standard matrix completion implementation besides the minimization of the weighted Frobenius norm. We then provide a short description of necessary changes when operating on FHE data.

Under FHE, the input matrices $M$ and $W$ are both encrypted using FHE schemes. There are still a few concerns we must look out for, notably non-addition, non-multiplication operations, and error accumulation from repeated multiplication.

The FHE-incompatible operations in the algorithm 1 are clipping prediction values, division in computing the loss, and singular value decomposition (SVD). For clipping and division by encrypted values, we decrypt the necessary values before performing plaintext operations. In practice, this may be implemented by having the server send encrypted values to clients, decrypt the value on the client-side then send the decrypted, plaintext value back to the server to continue execution of the algorithm. We provide a FHE-compatible implementation of SVD algorithms later in algorithm 3. Note that SVD implementations in popular Python libraries such as NumPy and SciPy cannot be used because they are incompatible with FHE.

For error accumulation, we execute repeated decryption and re-encryption to reset the accumulated error on values in encrypted matrices. During every step where we compute $X$ and $Y$ (which are encrypted), we decrypt and re-encrypt $X$ and $Y$ before proceeding with algorithm 1. This prevents loss of precision from multiplication and ensures the resulting predictions are accurate.

3.2. Robust Weight Precomputation

In this section, we propose algorithm 2 to compute the weights matrix $W$ used in semi-random matrix completion. This is a pre-processing step to algorithm 1. Our weight computation algorithm differs from previous work like Ge and Cheng in that we use a gradient descent method. During each epoch, we minimize the characteristic function $f = v^T(W - L)v$, where $L$ is the all 1’s matrix and $v$ is a unit eigenvector of $W - L$ corresponding to the largest eigenvalue. When updating values in the weights matrix $W$, we restrict ourselves to only updating entry $(i,j)$ of the weights matrix $W$ if $M_{i,j}$ is a revealed entry.

If the input matrix $M$ is non-square, it suffices to replace $W$ and $L$ with the appropriate block matrices of

$$
\begin{bmatrix}
0 & W \\
W^T & 0
\end{bmatrix}
$$

and
Under FHE, the input matrix $B$ and output matrix $W$ to algorithm 2 are both encrypted. To ensure compatibility with FHE, we again must watch out for incompatible operations or operations that result in extensive ciphertext multiplications. The only non-addition, non-multiplication computations needed are computing $v$, a unit eigenvector of $W - L$. We provide a FHE-compatible implementation of an eigenvalue extraction in algorithm 4. For ciphertext multiplications, it suffices to decrypt the unit eigenvector $v$ before performing computation, as this avoid accumulating multiplications of the encrypted weights matrix $W$ with other encrypted values.

### 3.3. Singular Value Decomposition on Encrypted Data

In this section, we provide an implementation for an FHE-compatible SVD, algorithm 3. This SVD implementation is used as a pre-processing step in semi-random matrix completion before the gradient descent subroutine is run. Our SVD implementation only computes the vectors corresponding to singular values, as we do not need the singular values themselves for matrix completion. Algorithm 3, like robust weight computation, also makes use of an FHE-compatible eigenvector extraction algorithm, algorithm 4. Note that we do not provide a plaintext SVD implementation, as existing SVD implementations in libraries like NumPy and SciPy are compatible with plaintext data.

### 3.4. Eigenvector Extraction on Encrypted Data

In this section, we present our algorithm for eigenvector extraction on FHE data, algorithm 4. Algorithm 4 is used as a subroutine by both our FHE-compatible SVD (algorithm 3) and robust weight computation (algorithm 2). Like the FHE-compatible SVD (algorithm 3), a non-FHE compatible version is not needed, as existing libraries like NumPy or SciPy already provide eigenvector extraction algorithms.

### 4. Evaluation

To evaluate the performance and feasibility of our secure, robust semi-random matrix completion implementation, we run benchmarks on a local machine with an 8-core Intel CPU (i7-10870H) and 16 GB of RAM. This is sufficient for benchmarking with plaintext data, but is insufficient for anything but the smallest matrices under FHE, as our implementation under FHE is most bottlenecked by RAM. Therefore, we perform most evaluations on plaintext data, as any trends regarding performance of robust semi-random matrix completion should generalize to the FHE case, albeit with worsened performance. In our evaluation, we aim to answer the following questions:

1. How good are the robust weights at sparsifying the input matrix, and do these weights produce better recommendations?
2. How long does weight computation take, and how well does it scale in regard to size of matrices?

All figures illustrating the performance of the robust matrix completion algorithms can be found in section ??.

#### 4.1. Accuracy

To evaluate the accuracy of the robust weights at sparsifying the input matrix, we evaluate the Frobenius norm of the robust weights compared to the all 1’s matrix, and the Frobenius norm of the ground truth matrix compared to the all 1’s matrix. The ground truth norms can be seen in figure 1(a), and the robust weighted norms can be seen in figure 1(b). Loss curves for robust weight computation can be seen in figure 1(e). Our robust weight computation produces weights approximately as spectrally similar to the all 1’s matrix as the ground truth of revealed entries. The spectral similarity of our computed weights and the ground truths grows as size of the matrix increases. For loss, we see that loss decreases as $\epsilon$ increases and more entries are revealed.

To evaluate the accuracy of the resulting recommendations, we compute a ground truth matrix, reveal some data IID with probability 0.2, and reveal an additional $\epsilon$ of the data from the 1st quadrant. We then run standard matrix completion and robust matrix completion on the noised data. We compare the Frobenius norm of the ground truth matrix with the non-robustly completed matrix and the robustly completed matrix in figure 1(c) and figure 1(d), respectively. Loss curves for matrix completion for the non-robust and robust implementations can be seen in figure 1(f). Robust matrix completion actually has worse accuracy than standard matrix completion when run on sufficiently small matrices (such as $50 \times 50$).
However, as matrix size and epsilon increases, accuracy of standard matrix completion declines much faster than its robust counterpart. Standard matrix completion in particular is adversely affected by increased values of $\epsilon$, whereas robust matrix completion has little change. For sufficiently large matrices with sufficient levels of corruption, our robust semi-random matrix completion algorithm outperforms standard matrix completion.

In addition, we attempted to visualize some of the weights to determine how well the semi-robust solution was learning the distribution of true values - as we can see from the given sample result in Figure 1(g) and 1(l), the weights learnt are quite reasonable. This is a more visual demonstration of confidence in the algorithm.

(a) Frobenius norms of ground truth matrices of varying sizes and $\epsilon$-error. Entries are revealed IID with probability 0.2

(b) Frobenius norms of robust weights of varying sizes and $\epsilon$-error. Entries are revealed IID with probability 0.2, then an additional $\epsilon$ portion of elements in quadrant 1 are revealed

(c) Frobenius norms comparing standard matrix completion with ground truth matrix of varying sizes and $\epsilon$-error. Entries are revealed IID with probability 0.2

(d) Frobenius norms comparing robust semi-random matrix completion with ground truth matrix of varying sizes and $\epsilon$-error. Entries are revealed IID with probability 0.2, then an additional $\epsilon$ portion of elements in quadrant 1 are revealed

(e) Loss curves for robust weight computations on a $250 \times 250$ matrix

(f) Loss curves for matrix completion on a $500 \times 500$ matrix comparing robust matrix completion with non-robust matrix completion
4.2. Performance

We evaluate the performance of our robust weight computation algorithm by measuring its runtime on matrices of various size and various levels of epsilon corruption. We only measure the runtime of the weight pre-processing step rather than the entire robust matrix completion algorithm. Runtime data can be found in figure 1(i). Our robust weight computation has minimal overhead that scales linearly with total entries in the matrix.

4.3. Secure Implementation

We also evaluate how the performance and accuracy of the robust weight algorithm change when operating over FHE data. FHE incurs significant performance cost, most notably in memory, so we are only able to run performance benchmarks for FHE tests on very small matrices. We evaluate the performance in the FHE case similarly to the plaintext case by assessing the runtime of the algorithm on matrices of varying size and epsilon corruption. These results can be found in figure 1(j). FHE drastically slows down performance. It has similar rate to robust matrix completion on plaintext, but the runtime problem is exacerbated by poor performance on small matrices.

We also assess the accuracy of the computed weights in a similar fashion to the plaintext weights. We evaluate the Frobenius norm of the robust and ground truth matrices to the all 1’s matrix. These results can be seen in figure 1(k) and figure 1(l). We see that FHE does not substantially impact the accuracy on weight computation. The main impact of FHE is on runtime and memory pressure rather than algorithmic correctness.

### Table 1

<table>
<thead>
<tr>
<th>Shape</th>
<th>Epsilon</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 × 50</td>
<td>0.05</td>
<td>326.787</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>308.572</td>
</tr>
<tr>
<td>250 × 250</td>
<td>0.669</td>
<td>859.180</td>
</tr>
<tr>
<td></td>
<td>0.584</td>
<td>946.772</td>
</tr>
<tr>
<td>500 × 500</td>
<td>2.063</td>
<td>2.108</td>
</tr>
<tr>
<td></td>
<td>2.171</td>
<td>2.171</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Shape</th>
<th>Epsilon</th>
<th>Shape</th>
<th>Epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 × 5</td>
<td>0.05</td>
<td>50 × 50</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>50 × 50</td>
<td>0.10</td>
</tr>
<tr>
<td>10 × 10</td>
<td>0.584</td>
<td>250 × 250</td>
<td>0.584</td>
</tr>
<tr>
<td></td>
<td>0.6501</td>
<td>250 × 250</td>
<td>0.6501</td>
</tr>
</tbody>
</table>
5. Conclusion

In this paper, we present a series of algorithms for implementing privacy-compliant robust matrix completion using FHE. We integrate these algorithms into a privacy-compliant recommendation service, SecRec, and compare it with standard matrix completion algorithms. We find that in the presence of adversarial data corruption, robust matrix completion results in more accurate recommendations at the cost of some runtime overhead.

In the plaintext setting, our robust precomputation algorithm does not incur much additional computation time. However, the performance of robust weight precomputation in the FHE setting is poor and scales poorly relative to the plaintext implementation. This is a current limitation of existing FHE schemes. Ultimately, we demonstrate the feasibility of combining robust machine learning algorithms with privacy-preserving techniques. Based on our performance results in FHE testing, future work should prioritize performance optimizations for robust matrix completion and FHE operations. Other avenues for additional developments include implementing robust algorithms in the FHE setting for when individual data points are noised, such as a FHE-compatible nuclear norm operation.

References


A. Appendix.

A.1. Algorithms

This section contains all additional algorithms presented throughout this paper.

**Algorithm 2 Weight Computation**

**Input:** $B$, a boolean matrix with 1’s corresponding to revealed entries in $M$, and 0’s otherwise, number of epochs $n_e$, number of sub-epochs, $n_s$, learning rate $\alpha$

**Notation Remark 1:** $a \leftarrow b$ means $a$ is updated to $b$

**Variables:** $W$, the weights matrix we want to solve for

1) **Setup**

Initialize matrix $W$ to the same size as $B$ with random values from $[0, 1]$

If $B_{i,j} = 0$, set $W_{i,j}$ to 0

Set $L$ to the same size as $B$ with all 1’s

If $W$ nonsquare, compute block matrices for $W$ and $L$ outlined above for $e \in 1...n_e$ do

Compute $v$, the unit eigenvector of $W - L$ corresponding to the largest eigenvalue

for $s \in 1...s_e$ do

Compute loss $\ell \leftarrow v^T(W - L)v$

Compute gradient matrix $G = v \otimes v$

Using row broadcast, set $G_{i,j} \leftarrow G_{i,j} \cdot B_{i,j}$ (use appropriate block matrix if $B$ non-square)

If $\ell < 0$, set $G \leftarrow -G$

Update weights matrix $W \leftarrow W - \alpha \cdot G$

end for

end for

Return Weights matrix $W$

If $W$ represents a block matrix, the returned weights $W$ can be easily extracted

**Algorithm 3 FHE-Compliant Singular Value Decomposition**

**Input:** $M \in \mathbb{R}^{n \times m}$ encrypted matrix, $r$ rank of matrix or maximum singular values to compute

**Notation Remark 1:** $a \leftarrow b$ means $a$ is updated to $b$

**Variables:** $W$, the weights matrix we want to solve for

**Variables:** $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{n \times r}$

for $r_i \in 1...r$ do

if $n > m$ then

$v \leftarrow$ unit eigenvector of $M$ corresponding to largest eigenvalue of $M$

Decrypt $v$ if $v$ not in plaintext

$u \leftarrow M \cdot v$

Decrypt and normalize $u$ on the client side

else

$u \leftarrow$ unit eigenvector of $M$ corresponding to largest eigenvalue of $M$

Decrypt $u$ if $u$ not in plaintext

$v \leftarrow M^T \cdot u$

Decrypt and normalize $v$ on the client side

end if

Set column $r_i$ of $U$ to $u$

Set column $r_i$ of $V$ to $v$

end for

$U \leftarrow $ encrypt $U$

$V \leftarrow $ encrypt $V$

Return $U^T$, $V$
Algorithm 4 FHE-Compliant Eigenvector Extraction

Input: $M \in R^{n \times m}$ encrypted matrix, $\epsilon$ error threshold, $m$ maximum iteration bound

Notation Remark 1: $a \leftarrow b$ means $a$ is updated to $b$

Variables: $v$, the eigenvector corresponding to the largest eigenvalue of $M$ we want to extract

if $n > m$ then
    Set $M \leftarrow M^T \cdot M$
end if

if $m > n$ then
    Set $M \leftarrow M \cdot M^T$
end if

Set $v \leftarrow$ a random plaintext vector

for $m_i \in 1...m$ do
    Set $v' \leftarrow M \cdot v$
    Decrypt and normalize $v'$ on the client side
    If $v' \cdot v > 1 - \epsilon$, set $v \leftarrow v'$ and break
    $v' = v$
end for

Return $v$