Privacy Preserving Collaborative Filtering

Emily Mu, Christopher Shao, Vivek Miglani

May 2017

1 Abstract

As machine learning and data mining techniques continue to grow in popularity, it has become ever more important to develop methods that preserve user security. One important problem in the field of machine learning is recommendation, which seeks to predict user ratings for unrated items using previous rating information. We focus specifically on the approach of similarity based collaborative filtering (CF) for recommendation, which involves computing similarity coefficients between items in order to extrapolate ratings for unrated items. We analyze an existing protocol for accomplishing privacy preserving collaborative filtering and identify potential vulnerabilities caused by a malicious server. We propose a modified protocol for privacy preserving collaborative filtering which eliminates the identified vulnerabilities. We implement the proposed system and evaluate the tradeoffs between the two systems in terms of performance and security.

2 Introduction

Privacy preserving machine learning algorithms have gained significant popularity in recent years. Machine learning algorithms benefit significantly from larger amounts of training data. Many parties may benefit from combining their data to train an improved model, but all parties want to ensure that their data is not revealed to the other parties. Privacy preserving ML algorithms focus on determining trained model parameters and allow prediction without any party revealing their raw data to the other parties.

A common problem within machine learning is the task of recommendation. The problem is generally formulated as follows. Take some set of users labeled $u_1, u_2, u_3, ...u_n$ and a set of items labeled $i_1, i_2, ..., i_m$. Known data includes ratings for some item-user pairs and sometimes also includes features of each item. Generally, we assume that the known data is a sparse $n * m$ matrix.

The goal is to predict ratings for the remaining user-item pairs, in order to recommend new items to users which will likely be rated highly by the user.

Two common classes of recommendation problems are collaborative filtering and content based filtering [1]. Collaborative filtering utilizes only the known ratings for user-item pairs, while content-based filtering takes into account not only the known ratings but also features representing each item.

A common example of the recommendation problem is that of movie recommendations. Consider a service, such as Netflix, in which users rate movies they have seen before. Using this data, the service desires to learn patterns from the known data and recommend unrated movies to each user. A collaborative filtering approach to this problem would simply use known user-movie ratings to make recommendations, while a content based filtering would utilize information regarding each movie (such as genre, leading actor, etc.) in addition to known user-movie ratings to make predictions.

Several methods have been utilized to approach recommendation problems in practice. These methods assume that a single party holds all the known user-item ratings and can use the data to learn the necessary information. We will recap the common machine learning algorithms utilized for this problem before exploring the privacy-preserving variant of this problem.

One common technique used to perform collaborative filtering is low rank matrix factorization [2]. Low rank matrix factorization utilizes known ratings to learn k-dimensional vectors to represent users and items. The predicted rating for a specific user and item is simply the dot product of the corresponding user vector and item vector. Let U represent the matrix with rows corresponding to each user vector, and let V represent the matrix with rows corresponding to each item vector. Based on this formulation, the product UV^T is a complete $n*m$ matrix containing predicted ratings for all user-item pairs. The vectors are often learned by performing alternating minimization on the following objective function:

$$J(U,V) = \frac{1}{n}\sum_{d\in D}(Y_d - UV_d^T) + \lambda(UU^T + VV^T)$$

where λ represents a regularization parameter and D represents all values in original matrix Y that are previously known.

Another common set of techniques for collaborative filtering are similarity based approaches [1]. These approaches use known user-item ratings to compute the cosine similarity (or other similarity metric) between pairs of users or pairs of items. These similarities are then utilized to extrapolate unknown ratings. We can define this approach formally for item-based similarity as follows.

Let $s(i_j, i_k)$ represent the similarity between two items. Then, we may predict the recommendation for an item with the formula:

$$P_{i,k} = R_k + \frac{\sum_{j=1}^{m}(r_{i,j} - R_j) * s(i_j, i_k)}{\sum_{j=1}^{m} s(i_j, i_k)}$$

where $r_{i,j}$ represents the rating vector of a given user u_i and R_k represents the average rating of item k.

In this work, we will explore a protocol for privacy preserving collaborative filtering using the similarity based approach.

2.1 Privacy Preserving Setup

We will now describe the setup for the privacy preserving collaborative filtering problem. As in the original setup, we have n users, labeled as $u_1, u_2, u_3, ...u_n$, who each have rated some products and would like to receive recommendations for unrated products. We also have a server, which is trusted with computing and storing information necessary to make recommendations, specifically the similarity between items. Note specifically that the users do not trust the server with their personal ratings and are only acceptable with the server learning anonymized patterns such as average rating and item similarity. Thus, users will only send encrypted versions of their rating information but still expect to receive accurate recommendations based on the rated items.

3 Protocol Description

3.1 Original Protocol

In order to accomplish the privacy-preserving collaborative filtering task, we take an approach using homomorphic encryption, based on the work of Badsha et al [1]. We will first describe the protocol defined by Badsha and then analyze potential vulnerabilities and modifications.

The protocol utilizes the El Gamal encryption scheme, which is defined as follows:

3.1.1 El Gamal Encryption Scheme

Key Generation:

Let G be a cyclic group with order q and generator g. A user will choose a random value $x \in \{1, ...q-1\}$. The user will then compute $h = g^x$ and publish $\{G, q, g, h\}$ as their public key. x will be this user's secret key.

Encryption:

Messages can be encrypted as follows. $y \in \{1, ...q-1\}$ is selected. The encryption can then be represented by

$$(C_1, C_2) = (g^y, m * h^y)$$

Decryption:

The user can decrypt messages with their private key x.

$$\frac{C_2}{C_1^x} = \frac{m * h^y}{(g^y)^x} = \frac{m * (g^x)^y}{(g^y)^x} = m$$

Homomorphic Properties:

Given the encryption of two messages $E(M_1) = (C_{11}, C_{12})$ and $E(M_2) = (C_{21}, C_{22})$, a valid encryption for $M_1 M_2$ can be computed by multiplying the encryptions for each of the messages.

$$E(M_1)E(M_2) = (C_{11}C_{21}, C_{12}C_{22}) = (g^{y_1} g^{y_2}, M_1 M_2 * h^{y_1} h^{y_2})$$

The main protocol involves two main parts, first computing the user averages of item ratings and the similarity matrix between items using user ratings and then providing recommendations using the calculated predicted ratings based on provided encrypted ratings.

The first phase, calculating user averages and the similarity matrix, can be computed as follows:

3.1.2 Computing Averages

1. Let all users $u_i = \{u_1, ...u_n\}$ choose public and private El Gamal keys y_i and x_i respectively. All users send their public keys y_i to the server and the server computes common public key $Y = \prod_{i=1}^{n} y_i$ which is broadcasted to all of the users to encrypt their ratings.

2. Let $r_{i,j}$ and $f_{i,j}$ represent the ratings and flags for user u_i for all items $i_j \in \{i_1, ...i_m\}$. The ratings are defined as previously. The flags are binary $f_{i,j} \in \{0, 1\}$ and is equal to 1 if a user has rated a product. Each user encrypts their ratings and flags as follows:

   $$M_i = \{E(g^{r_{i,j}}), E(g^{f_{i,j}})\}_{j=\{1,m\}}$$

   and sends these encryptions to the server.

3. The server then calculates the addition of all of the ratings and flags from the user encryptions using El Gamal's homomorphic encryption property.

$$(A_{1,j}, B_{1,j}) = \prod_{i=1}^{n} E(g^{r_{i,j}})$$

$$(A_{2,j}, B_{2,j}) = \prod_{i=1}^{n} E(g^{f_{i,j}})$$

where $(A_{1,j}, B_{1,j})$ and $(A_{2,j}, B_{2,j})$ represent the encryptions of the sums of the ratings and flags respectively for i_j. The server then broadcasts $(A_{1,j}, A_{2,j})_{j=\{1,m\}}$ to all of the users.

4. Each user u_i then computes the following message with their own private key

$$M_i^2 = (A_{1,j}^{x_i}, A_{2,j}^{x_i})_{j=\{1,m\}}$$

and sends this message back to the server.

5. The server can then decrypt the ciphertexts with these new messages as follows

$$C_{1,j} = \frac{B_{1,j}}{\prod_{i=1}^{n}(A_{1,j})^{x_i}}$$
$$C_{2,j} = \frac{B_{2,j}}{\prod_{i=1}^{n}(A_{2,j})^{x_i}}$$

Finally, the sum of the ratings and flags for item i_j (denoted by S_{r_j} and S_{f_j}) are computed by the server as follows

$$S_{r_j} = \log_g(C_{1,j})$$
$$S_{f_j} = \log_g(C_{2,j})$$

and the average rating of item i_j (R_j) can be computed by

$$R_j = \frac{S_{r_j}}{S_{f_j}}$$

Note that the server calculations for S_{r_j} and S_{f_j} do require calculating the log. However since these values are bounded, this log problem can be checked within the bounds in $O(m)$ time and does not require fully solving the discrete log problem.

6. The server then stores these R_j values in its database and is assumed to not reveal R_j or S_{r_j} and S_{f_j}.

3.1.3 Computing Similarity

1. Let all users $u_i = \{u_1, ...u_n\}$ compute $g^{r_{i,j} * r_{i,k}}$ and $g^{r_{i,j}^2}$ for all $i_j, i_k \in \{i_1, ...i_m\}$. Each user will then send the ciphertexts

$$M_i = \{E(g^{r_{i,j} * r_{i,k}}), E(g^{r_{i,j}^2})\}_{j,k \in \{1,...m\}; k \geq j}$$

to the server.

2. The server then calculates the homomorphic product of all of the users' ciphertexts using El Gamal's homomorphic encryption property.

$$(A_{j,k}, B_{j,k}) = \prod_{i=1}^{n} E(g^{r_{i,j} * r_{i,k}})$$

$$(A_j, B_j) = \prod_{i=1}^{n} E(g^{r_{i,j}^2})$$

The server then broadcasts $(A_j, A_{j,k})_{j,k \in \{1,...m\}; k \geq j}$ to all users.

3. Each user u_i then computes the following message with their own private key

$$M_i^2 = (A_{j,k}^{x_i}, A_j^{x_i})_{j,k \in \{1,...m\}; k \geq j}$$

and sends this message back to the server.

4. The server then can decrpyt the ciphertexts with these new messages as follows

$$C_{j,k} = \frac{B_{j,k}}{\prod_{i=1}^{n} (A_{j,k})^{x_i}}$$

$$C_j = \frac{B_j}{\prod_{i=1}^{n} (A_j)^{x_i}}$$

Finally, the pairwise products and squares of ratings for item i_j (denoted by $S_{j,k}$ and S_j) are computed by the server as follows

$$S_{j,k} = \log_g C_{j,k}$$
$$S_j = \log_g C_j$$

The server can then compute the similarity between any two items i_j and i_k

$$s(i_j, i_k) = \frac{S_{j,k}}{\sqrt{S_j}\sqrt{S_k}}$$

Note that the server calculations for $S_{j,k}$ and S_j do require calculating the log. However since these values are bounded, this log problem can be checked within the bounds in $O(m^2)$ time and does not require fully solving the discrete log problem.

5. The server then stores these $s(i_j, i_k)$ values in its database and is assumed to not reveal $s(i_j, i_k)$ or $S_{j,k}$ and S_j.

The second phase, calculating predicted ratings using the similarity matrix, average ratings, and encrypted ratings is accomplished by the following steps.

3.1.4 Collaborative Filtering Based Recommendations

1. For all items $i_j = i_1, ...i_m$, target user u_i sends the encrpyted ciphertexts

$$M_i = \{E(g^{r_{i,j}})\}_{j=1,...m}$$

to the server.

2. The server generates the ciphertexts of the numerator. $R_k * \sum_{j=1}^{m} s(i_k, i_j) + \sum_{j=1}^{m}(r_{i,j} - R_j) * s(i_k, i_j)$. It encrypts

$$E(g^{R_k * \prod_{j=1}^{m} s(i_k, i_j)})$$

$$E(g^{R_j})$$

The ciphertexts of the numerator is generated by

$$(A_{4,k}, B_{4,k}) = E(g^{R_k * \prod_{j=1}^{m} s(i_k, i_j)}) * (\prod_{j=1}^{m} \frac{E(g^{r_{i,j}})}{E(g^{R_j})}^{s(i_k, i_j)})$$

$$k = \{1, ...m\}$$

3. The server generates the ciphertexts of the denominator by encrypting

$$(A_{5,k}, B_{5,k}) = E(g^{\sum_{j=1}^{m} s(i_k, i_j)})$$

4. The server sends the following messages to user u_i

$$\{A_{4,k}, B_{4,k}, A_{5,k}, B_{5,k}\}_{k=\{1,...m\}}$$

5. The user u_i decrypts the ciphertexts by using their own private key x_i

$$C_{6,k} = \frac{B_{4,k}}{(A_{4,k})^{x_i}}$$
$$C_{7,k} = \frac{B_{5,k}}{(A_{5,k})^{x_i}}$$

The user can then compute

$$n_k = \log_g C_{6,k}$$
$$d_k = \log_g C_{7,k}$$

and calculate the rating prediction as follows

$$P_{i,k} = \frac{n_k}{d_k}$$

Note that the user calculations for n_k and d_k do require calculating the log. However since these values are bounded, this log problem can be checked within the bounds in $O(m)$ time and does not require fully solving the discrete log problem.

3.2 Potential Vulnerabilities

One key vulnerability in the protocol described above is potential malicious behavior by the semi-trusted server. The semi-trusted server may have incentive to deviate from the protocol in order to obtain a user's personal recommendations, which are expected to remain secure in this procedure. The original proposal assumes that the server will not collude to try to obtain a user's ratings. But in reality, the central server, often a business, may have financial incentive to obtain a individual's ratings and may deviate from the protocol to accomplish this.

The semi-trusted server can accomplish a simple attack by deviating from the protocol in step 3 when computing the average. The server is required to broadcast the product of all the individual encryptions to the users and each user decrypts this product with their private key. If, instead of sending the product of the individual encryptions, the server sends the user the first part of his or her own encrypted ratings and flags (g^y), the value returned by the user would be h^y, where y is the randomly chosen value used to encrypt the ratings. The server can then easily decrypt the user's rating by dividing the encrypted ratings by h^y.

One modification to the protocol which can mitigate this issue is requiring the server to maintain a public ledger containing the encryption of each user's ratings and flags. Maintaining this public ledger would allow any user to verify

that the value being sent for decryption correctly corresponds to the product of the individual encryptions.

Unfortunately, this simple modification is not sufficient to inhibit a malicious server from deceiving a user to decrypt their own rating. The server can accomplish this by adding another "fake" user, and choosing the user's encrypted ratings appropriately to deceive a specific user into providing their ratings.

More formally, the attack could be accomplished as follows. The server, in step 2 of the average computation, receives the encryptions of g^{r_i} from each user. These encryptions are in the form:

$$(C_{i,j,1}, C_{i,j,2}) = E(g^{r_{i,j}})$$

where $C_{i,j,2} = (C_{i,j,1})^{x_i} g^{r_{i,j}}$. The server computes the product:

$$A_{1,j} = \prod_{i=1}^{n} C_{i,j,1}$$

Suppose the server either impersonates or controls the identity of user n. Instead of following the standard encryption scheme, the server could choose the value of $C_{i,j,1}$ to be equal to the multiplicative inverse (mod q) of the product of $C_{i,j,1}$ of user 2 through n - 1.

$$C_{n,j,1} = (\prod_{i=2}^{n-1} C_{i,j,1})^{-1} \textbf{ mod q}$$

Note that this multiplicative inverse can be computed efficiently using the extended Euclid's algorithm, since q is known. The value of $A_{1,j}$ would then be:

$$A_{1,j} = \prod_{i=1}^{n} C_{i,j,1} = C_{1,j,1} C_{n,j,1} \prod_{i=2}^{n-1} C_{i,j,1}$$

$$A_{1,j} = C_{1,j,1} (\prod_{i=2}^{n-1} C_{i,j,1})^{-1} \prod_{i=2}^{n-1} C_{i,j,1}$$

$$A_{1,j} = C_{1,j,1}$$

The product is simply equal to $C_{1,j,1}$ mod q. When user 1 decrypts this product, the server will receive the value of $(C_{1,j,1})^{x_i}$ and can easily compute

$$\frac{C_{i,j,2}}{(C_{1,j,1})^{x_1}} = \frac{(C_{i,j,1})^{x_1} g^{r_1,j}}{(C_{1,j,1})^{x_1}} = g^{r_1,j}$$

This clearly provides the server plaintext access to user 1's rating $r_{1,j}$.

Thus, even if the user verifies that the product is computed correctly, the server can still obtain the user's rating by simply choosing the encryption for one specific user.

3.3 Updated Protocol

To prevent the potential vulnerabilities identified in section 3.2, we propose the following variant of the original protocol. This modified protocol no longer assumes that the server does not deviate from the protocol to obtain an individual user's rating and thus protects users from the potential attacks by a malicious server described above.

The modified protocol utilizes a hash function H, which is assumed to be one-way and collision-resistant.

The process of computing averages now proceeds as follows. Note that steps 1 and 2 remain unchanged from the original protocol.

1. Let all users $u_i = \{u_1, ...u_n\}$ choose public and private El Gamal keys y_i and x_i respectively. All users send their public keys y_i to the server and the server computes common public key $Y = \prod_{i=1}^{n} y_i$ which is broadcasted to all of the users to encrypt their ratings.

2. Let $r_{i,j}$ and $f_{i,j}$ represent the ratings and flags for user u_i for all items $i_j \in \{i_1, ...i_m\}$. The ratings are defined as previously. The flags are binary $f_{i,j} \in \{0, 1\}$ and is equal to 1 if a user has rated a product. Each user encrypts their ratings and flags as follows:

$$M_i = \{E(g^{r_{i,j}}), E(g^{f_{i,j}})\}_{j=\{1,m\}}$$

3. Each user i then computes $H(M_i)$ and sends this value to the server. Sending this hash in advance serves as a form of a commitment, ensuring that no user changes their encryption upon seeing the encrypted ratings of other users.

4. The server posts the values of $H(M_i)$ for all users on a public ledger, accessible to all users.

5. Once $H(M_i)$ for all users has been posted to the public ledger, each user sends M_i to the server. Again, the server posts all values of M_i to the public ledger accessible to all other users.

6. The server then calculates the addition of all of the ratings and flags from the user encryptions using El Gamal's homomorphic encryption property.

$$(A_{1,j}, B_{1,j}) = \prod_{i=1}^{n} E(g^{r_{i,j}})$$

$$(A_{2,j}, B_{2,j}) = \prod_{i=1}^{n} E(g^{f_{i,j}})$$

where $(A_{1,j}, B_{1,j})$ and $(A_{2,j}, B_{2,j})$ represent the encryptions of the sums of the ratings and flags respectively for i_j. The server then broadcasts $(A_{1,j}, A_{2,j})_{j=\{1,m\}}$ to all of the users.

7. Each user u_i then computes the following message with their own private key

$$M_i^2 = (A_{1,j}^{x_i}, A_{2,j}^{x_i})_{j=\{1,m\}}$$

Before sending this information back to the server, the user must verify the following properties to ensure that the server has not acted maliciously:

(a) For each user a, computing the hash of M_a must equal to $H(M_a)$ which was set prior to the user revealing M_i.

(b) The products $(A_{1,j}, A_{2,j})_{j=\{1,m\}}$ must be equal to $\prod_{i=1}^{n} E(g^{r_{i,j}})$ and $\prod_{i=1}^{n} E(g^{f_{i,j}})$. A user can verify this since each M_a is available in the public ledger.

Once these properties are verified, the user sends the value of M_i^2 to the server.

8. Steps 5 and 6 of the original protocol remain unchanged, allowing the server to obtain the average rating for each item.

The procedure for computing similarity is also modified analogously. Users will first send the hash of their encryptions to the server, to be included in the public ledger. Once all hashes are publicly available, each user sends their encryption to the server, which is also added to the public ledger. Before decrypting the product, each user verifies that the product has been computed accurately and that each user's pre-transmitted hash value matches the hash of the encryption.

4 Results

To evaluate the feasibility and performance impact of the modified protocol, we implemented both the original and modified protocol using Python. [1]

Using our implementation, we obtained the following results comparing the performance of the original and modified protocols.

Testing was performed by randomly generating rating and flag information for the given number of users and items. To avoid the variable overhead of identifying a prime, all tests were performed using generator g = 2 and prime q = 1000003 for the El Gamal encryption scheme.

The total computation time (for all users), as a function of total number of users, is shown in Figure 1.

[1] Our source code is publicly available at https://github.mit.edu/vivekm/857Project

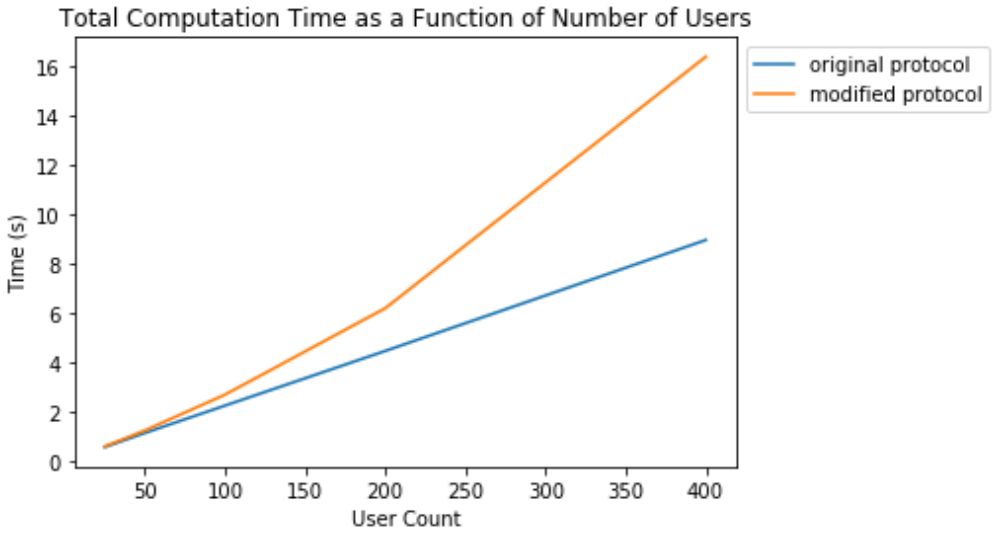

Figure 1: Computation Time vs Number of Users, Fixed Number of Items = 7

The total computation time (for all users), as a function of number of items, is shown in Figure 2.

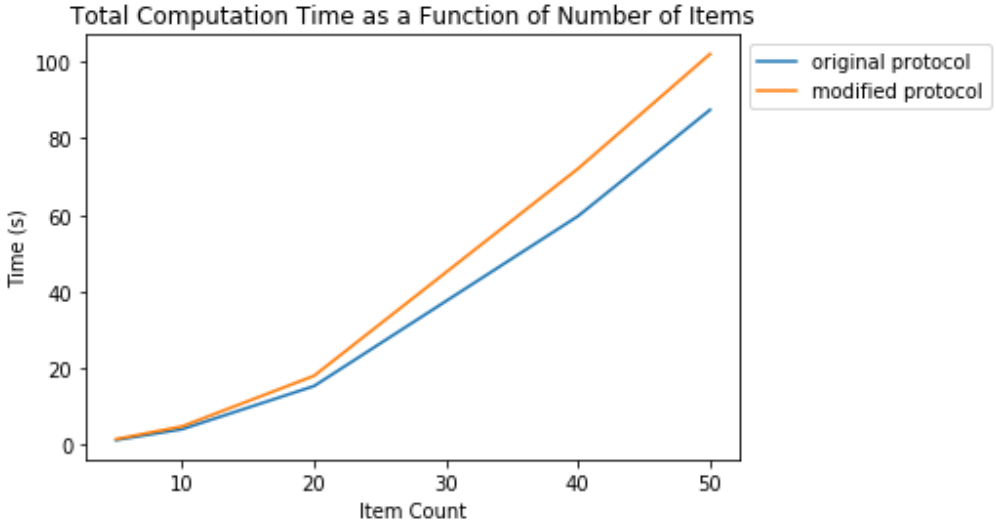

Figure 2: Computation Time vs Number of Items, Fixed Number of Users = 25

5 Evaluation

We evaluate the proposed, modified protocol in terms of the two main goals, security and performance.

5.1 Security

It is evident that the potential vulnerabilities described in section 3.2 are no longer possible based on the proposed modification.

The first attack, in which the server simply returns the user's encryption (or the encryption multiplied by E(1)) for decryption is not possible, since each user's encryption is available in the public ledger, and the user verifies that the product matches the value being decrypted.

The second attack, in which the server modifies or control one user's encryption is also no longer possible. This is because the hashes of each encryption are committed to in advance, prior to revealing the encryptions. By the one-wayness of the hash function, the server cannot determine any user's encryption and thus cannot use them in order to choose an encryption which is equal to the multiplicative inverse of a product of encryptions. Additionally, by the collision-resistance of the hash function, the server cannot change the controlled user's encryption after viewing the other user's encryptions, since this would require finding multiple encryptions with the same hash.

It is also necessary to ensure that the proposed modifications do not in any way compromise the security of the original protocol. The modified protocol publicizes the encrypted message of each user as well as the hash of the encryption of each user. Since the El Gamal encryption scheme is shown to be semantically secure, a polynomial-time bounded adversary cannot gain any information regarding user ratings from the encryptions or hashes in the public ledger.

5.2 Performance

The additional steps performed by the user in the modified proposal clearly add a significant computational cost. Each message M_i, is of length $O(m)$, where m is the number of items, since the message includes the encryption of the rating and flag for each item. Thus, verifying $H(M_i)$ for each user i takes $O(mn)$ time, where n is the number of users.

Additionally, each user must multiply the individual encryptions to verify the value provided by the server. Multiplying the n user encryptions takes $O(n)$ time and this process must be done for each of the m items, taking a total of $O(mn)$ time.

Note that in the original protocol, the computational cost for users is simply

$O(m)$, since the user needs to only encrypt each of their ratings (for each item). The computational cost for a single user did not depend on the total number of users in the original protocol.

In the modified protocol, the number of users is also a factor of a single user's computational cost. in a scenario with a large number of users, it is evident that the computational cost of the modified proposal would be significantly higher than that of the original proposal. This conclusion is reinforced by the empirical results of the implementation, as shown in section 4. In particular, the proportion of the total computation time taken up by users' verification (the difference in times between the original protocol and modified protocol) increases as the number of users increases. For example, in Figure 1, the proportion at 200 users is about 0.3, while the proportion at 400 users is about 0.5. This suggests that as the number of users grows even larger, the total computation time will be dominated by just verification of other users' hashes. As seen in Figure 2, the computation time of the modified protocol still increases when the number of users is fixed and the number of items increases. The difference is not as significant as in the previous case, since adding another item only requires each user to verify one more encryption hash, rather than verify another set of encryption hashes for a new user.

Thus, there exists a clear tradeoff in terms of security and performance, the achievement of greater security comes at the price of increased computational overhead for users, especially as the number of users increases.

6 Future Work

This section will discuss potential improvements to our existing algorithm and implementation and further directions of work.

6.1 Performance Improvements

As discussed in the previous section, there exists a tradeoff between improved security and improved performance. Although our modified protocol has significantly higher computational cost than the original protocol, several modifications can be made to improve its performance.

Since our implementation was completed in Python, several methods can be used to optimize performance. One simple method to improve performance may be to implement the algorithm a lower-level language. Another method may be to take advantage of potential steps that can be processed in parallel, for example, user encryption steps.

6.2 Protocol Improvements

Collaborative filtering inherently only takes into account item-based similarity to provide recommendations. One area of future work may be to provide a protocol to take advantage of user-based similarity to provide more accurate and detailed recommendations to users. However, since user-based similarity methods require more information about users, modifications to this protocol may be necessary to better protect user information.

7 Conclusion

Recommendation is an important and prevalent problem in the growing field of machine learning. Providing accurate and helpful recommendations to users affects the success of online commerce tools. In this paper, we seek to provide a privacy-preserving approach to the collaborative filtering recommendation method. We improve upon the algorithm proposed by Basha et al. [1] by placing less trust upon the server. Our evaluation demonstrates that our security improvements come at the cost of performance. Consequently, future directions of work may involve reworking the protocol to take advantage of user information or increasing performance to enhance usability of the current protocol.

References

[1] Badsha, Shahriar, Xun Yi, and Ibrahim Khalil. "A Practical Privacy-Preserving Recommender System." Data Science and Engineering 1.3 (2016): 161-77. Web.

[2] Nikolaenko, Valeria, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh. "Privacy-preserving matrix factorization". In ACM CCS 2013, pages 81–8121–812.

[3] Zhang, Sheng, James Ford, and Fillia Makedon. "Deriving Private Information from Randomly Perturbed Ratings." Proceedings of the 2006 SIAM International Conference on Data Mining (2006): 59-69. Web.

[4] Polat, H., and Wenliang Du. "Privacy-preserving Collaborative Filtering Using Randomized Perturbation Techniques." Third IEEE International Conference on Data Mining (n.d.): n. pag. Web.