pTwitterRec: A Privacy-Preserving Personalized Tweet Recommendation Framework

Bisheng Liu and Urs Hengartner
Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, Canada
{bisheng.liu, urs.hengartner}@uwaterloo.ca

ABSTRACT
Twitter is one of the most popular Online Social Networks (OSNs) nowadays. Twitter users retrieve information from other users by subscribing to their tweets. Twitter users, especially those who have many followees, may receive hundreds or even thousands of tweets daily. Currently, all tweets are shown to users in chronological order. Consequently, a Twitter user may accidentally overlook useful and interesting tweets because the user is overwhelmed by the huge volume of uninteresting tweets. Researchers in the recommendation system community have proposed using recommendation techniques such as collaborative filtering to predict users' preference of tweets and highlight those tweets in which users are most likely to be interested. At the same time, while OSNs such as Twitter have enabled people to conveniently share information and interact with each other online, OSN users are getting increasingly concerned about their online privacy. Researchers in the security community have proposed using techniques such as encrypted tweets to protect users' privacy. In this paper, we propose a privacy-preserving personalized tweet recommendation framework, *pTwitterRec*, in a Twitter-like social network where users' tweets are hidden from the OSN provider. *pTwitterRec* provides users with personalized tweet recommendations while keeping users' tweets and interests hidden from the OSN provider as well as other unauthorized entities. *pTwitterRec* splits the tweet recommendation task between the provider and a semi-trusted third party, so that neither can derive users' sensitive information alone while working together to provide users with personalized tweet recommendations. We implement a prototype and demonstrate through evaluation that *pTwitterRec* incurs tolerable overhead on today's smartphones.

Categories and Subject Descriptors
H.3.3 [**Information Search and Retrieval**]: Information filtering; J.4 [**Computer Applications**]: Social and behavioral sciences; K.4.1 [**Computers and Society**]: Public Policy Issues – Privacy

Keywords
Privacy Protection; Tweet Recommendations; Personalization

1. INTRODUCTION
Twitter is a popular online social networking website and microblogging service that allows users to conveniently share short messages of up to 140 characters, known as "tweets", with their online followers. With more than 550 million active registered Twitter users nowadays, approximately 58 million tweets are generated per day [38]. Twitter users on average follow 80 people [29], which leads to hundreds or even thousands of new tweets received by each user every day. Currently, all tweets are shown to users in chronological order, which may lead to users missing some informative and interesting tweets simply because users are overwhelmed by other tweets that do not interest them at all. Researchers [10, 14, 21] have proposed using various recommendation techniques that utilize all kinds of information available on Twitter (such as users' tweet history and social relations) to rank tweets and display those tweets that might interest the user most on top.

In the meantime, users of Online Social Networks (OSNs) such as Twitter are increasingly concerned about privacy issues [30]. In general, OSN providers are considered to be trustworthy, as they have a valuable reputation to maintain and any suspicion of malicious behaviors could potentially lead to a significant loss of users. As a result, users trust OSN providers by default to properly safeguard and manage the contents published by users. Unfortunately, users' privacy is at risk by placing trust completely on OSN providers. For example, OSNs' Terms of Service typically include clauses that explicitly/implicitly allow the provider to mine user content for the purpose of delivering targeted advertising or reselling to third party services [12]. Also, in regions where OSNs are closely monitored by unscrupulous and suppressive governments [32], it is crucial to guarantee that only users' explicitly authorized followers/friends can access their data. Researchers in the security community have proposed several approaches [3, 12, 16, 23] to achieve this goal by delegating control over content to end users.

The contributions of this paper are as follows: Our work is the first, to our best knowledge, to propose a privacy-preserving personalized tweet recommendation framework, *pTwitterRec*. We enhance an existing personalized tweet recommendation algorithm with a few cryptographic protocols and split the recommendation task between the provider of the Twitter-like service and a semi-trusted third party, so that users obtain personalized tweet recommendations without leaking any sensitive information to unauthorized entities. We demonstrate the practicality through implementation and evaluation of the overhead incurred by *pTwitterRec*. *pTwitterRec* only introduces minimum overhead on the user side while achieving the privacy goal, compared with the original tweet recommendation algorithm, which was designed with no privacy in mind. In this paper, we positively answer the following question: in a Twitter-like OSN, can we provide users with *the benefits of personalized tweet recommendations* while *keeping the contents of users' tweets and users' interests hidden from the provider as well as other unauthorized parties*?

The remainder of this paper is organized as follows: In Section 2, we overview the related work. In Section 3, we introduce the background of our work. In Section 4, we describe our system and threat models. In Section 5, we present in detail our proposed privacy-preserving personalized tweet recommendation framework *pTwitterRec*. In Section 6, we discuss the security and privacy aspects and a possible extension of *pTwitterRec*. We describe the server-side implementation in Section 7. We analyze and evaluate the overhead incurred by *pTwitterRec* in Section 8. Finally we conclude our work in Section 9.

2. RELATED WORK

2.1 Tweet Recommendations

Researchers have proposed recommendation systems for Twitter. Chen et al. [9] focus on recommending URLs on Twitter that a Twitter user might find interesting. Hannon et al. [17] develop a followee recommender system for Twitter that utilizes Twitter's real-time information as a source of recommendation knowledge. Diaz-Aviles et al. [13] propose using collaborative filtering to recommend hash tags on Twitter in real-time.

Some researchers focus on recommending tweets that are interesting to users. Duan et al. [14] propose using *learning to rank* algorithms to rank tweets based on the user query. Their approach considers quality of tweets and authority of the publishers but does not consider user personalization. Ramage et al. [31] propose using latent variable topic models like *Labeled LDA* to map the content of the Twitter feed into dimensions and then filter Twitter feeds for users. Their approach uses content-based filtering and does not consider users' social relations. Naveed et al. [21] propose using a learning approach based on pure content features to predict the probability of a message being retweeted. Their work does not consider user personalization, either. Chen et al. [10] propose using collaborative ranking to recommend tweets that are personally interesting to a user. Their approach considers not only explicit features such as quality of the tweet and authority of the publisher, but also tweet topic level latent factors and social relation latent factors. Experiments on real-world data show that their approach outperforms other tweet recommendation algorithms in terms of *mean average precision* (MAP).

Our work is based on the tweet recommendation algorithm proposed by Chen et al. [10]. In addition, we provide users with personalized tweet recommendations while preserving user privacy.

2.2 Privacy-preserving Recommendations

Privacy for traditional recommender systems such as collaborative filtering has been extensively studied. Canny [7, 8] propose using homomorphic encryption and a peer-to-peer protocol to provide privacy for model-based collaborative recommender systems. Polat and Du [25–28] propose that customers adopt randomized perturbation techniques to disturb their private data before sending the data to the service provider for recommender systems. Aïmeur et al. [1] propose using a semi-trusted third party to distill encoded sensitive customer information, which can be used to produce recommendations but cannot be decrypted by any of the concerned parties alone.

In personalized tweet recommendations, tweets are regarded as items and many unique Twitter features such as content information and social relation information need to be utilized to improve recommendation accuracy. None of the abovementioned approaches can be trivially adapted to offer personalized tweet recommendations while preserving user privacy.

2.3 Privacy-preserving OSNs

There has been significant work in OSN privacy. Our work is most related to approaches that aim to protect social network users' privacy from a curious OSN provider. Systems such as *FlyByNight* [22], *NOYB* [16] and *Facecloak* [23] store users' data with the untrusted provider but protect its content with encryption. Conti et al. [11] propose establishing a virtual private network between friends on OSNs to share sensitive information. Singh et al. [34] propose *Twitsper*, a wrapper around Twitter that enables private group communication among Twitter users while preserving Twitter's business interests. Feldman et al. [15] propose *Frientegrity*, a framework where an OSN provider observes only encrypted data and cannot deviate from correct execution without being detected. De Cristofaro et al. [12] propose *Hummingbird*, a variant of Twitter that protects users' tweets as well as users' interests from a semi-trusted OSN provider.

Our work, however, focuses primarily on protecting users' privacy from a curious OSN provider while offering users personalized tweet recommendations. Our privacy-preserving tweet recommendation framework *pTwitterRec* can be built upon any of the abovementioned approaches as long as the OSN provider is able to distribute tweets to the corresponding followers without learning the tweets.

3. BACKGROUND

3.1 Twitter

Twitter is one of the most popular OSNs that enable users to send and read *tweets*, which are text messages limited to 140 characters. A user receives tweets from other users by subscribing to their tweets. Twitter terminology relevant to this paper includes:

- *Publisher*: A user who posts a tweet.
- *Follower*: A user who follows others' tweets.
- *Followee*: A user whose tweets are being followed by others.
- *Tweets*: short messages posted by a user/publisher.
- *Retweets*: someone else's tweets that a user chooses to share with all of the user's followers.
- *Service Provider* (*SP*): the centralized entity of Twitter that maintains all user profiles and tweets, and distributes users' tweets to followers.

Tweets are public by default; that is, any registered user can read other users' public tweets. However, upon registration, a user has the option of keeping the tweets only visible to the user's approved Twitter followers, known as *protected tweets*. Nonetheless, all tweets published by users are not hidden from the SP.

3.2 Personalized Tweet Recommendations

Recommending useful tweets to a user is a challenging problem. The key of the problem is how to use any information available on Twitter to decide whether or not a user is interested in a tweet. Such information available on Twitter includes the quality of the tweet, the authority of the publisher, the user's previous tweets, etc. With such information as input, the SP adopts various machining learning techniques such as *Collaborative filtering* [4, 6], *LDA* [5] or *RankSVM* [19] to learn a model to predict users' interests in tweets. Among existing recommendation algorithms for social updates [5, 10, 14, 18, 19, 21, 35], we adopt the collaborative personalized tweet recommendation algorithm proposed by Chen et al. [10] as the building block of our privacy-preserving tweet recommendation framework *pTwitterRec*. We choose their algorithm over others because: ***a)*** their tweet recommendation model not only incorporates explicit tweet features such as the quality of the tweet and the authority of the publisher, but also uses topic level latent factors of tweets to capture users' interests and uses latent factors to model users' social relations; ***b)*** their model outperforms other up-to-date recommendation models with reasonable computation cost. For example, the empirical results show that their method achieves 46.84% and 17.41% improvements compared with the RankSVM method and joint matrix factorization method [35] in terms of Mean Average Precision (MAP).

We briefly describe their tweet recommendation algorithm. For a given user *u* and two received tweets *k* and *h*, assuming that the ranks of tweets *k* and *h* are r_k^u and r_h^u respectively (user *u* is more interested in tweet *k* than tweet *h* if r_k^u is larger/higher than r_h^u), the authors model the probability of the orders of tweets *k* and *h* for user *u* as follows:

$$P(r_k^u > r_h^u \mid u) = \frac{1}{1+e^{-(y_{u,k}-y_{u,h})}} \tag{1}$$

where $y_{u,k}$ represents the predicted rating of user *u* for tweet *k*.

Then, the authors define the rank preference set *D* as follows:

$$D = \{<u,k,h> \mid k \in Re(u), h \notin Re(u)\} \tag{2}$$

where $Re(u)$ represents the set of tweets user *u* has retweeted. *D* is built based on the assumption that *users are more interested in those tweets that they have retweeted than those they have not retweeted*. For a tuple <*u*, *k*, *h*> in *D*, an ideal rating prediction model $y_{u,k}$ maximizes the probability of the ranking order that user *u* ranks tweet *k* higher than tweet *h*. To solve this problem for all tuples in *D*, it is equivalent to solve the following objective by learning the optimal parameters for the rating prediction model $y_{u,k}$:

$$\min \sum_{<u,k,h \in D>} \ln(1+e^{-(y_{u,k}-y_{u,h})}) + \text{regularization} \tag{3}$$

Finally, the authors define the prediction model $y_{u,k}$ as follows:

$$y_{u,k} = \sum_{j\in F} b_j r_j^{u,k} + p_u^T\left(\frac{1}{Z^k}\sum_{w\in T_k} q_w + \alpha d_{p(k)}\right) \tag{4}$$

In the first part of equation 4, the authors use $\sum_{j\in F} b_j r_j^{u,k}$ to capture the impact of explicit features (such as the quality of the tweet and the authority of the publisher) on user *u*'s rating of tweet *k*, where $r_j^{u,k}$ is the *j*-th explicit feature computed from tweet *k* regarding user *u*, *F* is the set of explicit features and b_j is the system-wide weight corresponding to the *j*-th explicit feature, which is to be learned in the prediction model. The second part of equation 4 considers a latent factor model (described in Appendix A.1). In brief, the authors use $p_u^T \sum_{w\in T_k} q_w$ to capture the impact of words contained in tweet *k* on user *u*'s rating of the tweet, where p_u is the low dimensional representation of user *u* in the latent feature space R^d, T_k is the word set contained in tweet *k* and q_w represents the low dimensional representation of word *w* in the latent feature space R^d; the authors use $p_u^T d_{p(k)}$ to capture the impact of the social relation between user *u* and the publisher of tweet *k* on user *u*'s rating of the tweet, where $d_{p(k)}$ is the low dimensional representation of the publisher of tweet *k* in the latent feature space R^d. In equation 4, Z^k is the normalization term defined as the cardinality of the word set T_k, and α is a predefined system-wide parameter that indicates the importance of social relations relative to words contained in the tweet.

Using *D* as the training datasets, by solving objective 3, the SP learns the parameters of the tweet rating prediction model: the system-wide weight vector *b*, the latent factor for user *u* denoted as p_u, the latent factor for each word *w* denoted as q_w and the latent factor for the publisher of tweet *k* denoted as $d_{p(k)}$. Note that among all parameters to be learned in the prediction model, only the value of p_u depends on user *u* and all other parameters are global values. To solve objective 3, the authors perform stochastic gradient descent (as described in Section 3.3). After the prediction model $y_{u,k}$ is learned, the SP estimates a user's rating for a tweet using equation 4.

3.3 Stochastic Gradient Descent

Chen et al. [10] adopt stochastic gradient descent to solve objective 3. For each tuple <*u*, *k*, *h*> in *D*, the SP computes the descent of each parameter as follows and updates the parameters by moving in the direction of negative gradient:

$$\frac{\partial \ell}{\partial p_u} = \hat{e}\left(\frac{1}{Z^k}\sum_{w\in T_k} q_w - \frac{1}{Z^h}\sum_{w\in T_h} q_w + \alpha(d_{p(k)} - d_{p(h)})\right) - \lambda_1 p_u \tag{5}$$

$$\frac{\partial \ell}{\partial q_w^k} = \frac{1}{Z^k}\hat{e}p_u - \lambda_2 q_w^k, \frac{\partial \ell}{\partial q_w^h} = -\frac{1}{Z^h}\hat{e}p_u - \lambda_2 q_w^h \tag{6}$$

$$\frac{\partial \ell}{\partial d_{p(k)}} = \alpha\hat{e}p_u - \lambda_3 d_{p(k)}, \frac{\partial \ell}{\partial d_{p(h)}} = \alpha\hat{e}p_u - \lambda_3 d_{p(h)} \tag{7}$$

$$\frac{\partial \ell}{\partial b_j} = \hat{e}(r_j^{u,k} - r_j^{u,h}) - \lambda_4 b_j \tag{8}$$

$$\text{where } \hat{e} = 1 - \frac{1}{1+e^{-(y_{u,k}-y_{u,h})}} \tag{9}$$

Here, q_w^k represents the latent factor of word contained in tweet *k*. The algorithm loops over all tuples in *D* and updates the parameters accordingly.

4. SYSTEM AND THREAT MODEL

4.1 System Model

In *pTwitterRec*, we assume a privacy-preserving Twitter-like social network where tweets posted by users are hidden from the SP. We assume that users install a client application to participate in the social network. In the remainder of the paper, we refer to the client software installed on user *u*'s device as user *u* for simplicity. In *pTwitterRec*, upon registration, each user's account is protected by default; that is, a user's tweets are only visible to the user's explicitly authorized followers. In addition, the user's tweets are hidden from the SP as well. We assume that a user encrypts the tweets using a secret key (not known by the SP) shared only among the user's authorized followers before posting them to *pTwitterRec*. The SP is responsible for storing encrypted tweets published by users and delivering those tweets to their corresponding followers. How to construct *pTwitterRec* such that it meets the abovementioned assumptions is out of the scope of our work; as described in the related work, systems such as FlyByNight [22] and Hummingbird [12] can be adapted to implement *pTwitterRec*.

For tweet recommendations, to prevent the SP from learning a user's interests, in *pTwitterRec*, we introduce a semi-trusted third party server, denoted as the *word server* (WS). Users, the SP and the WS cooperate and learn the personalized tweet recommendation model. More specifically, users are mainly responsible for collecting and preparing training samples that are used to learn the recommendation model, the SP is responsible for computing the weight parameters for explicit features, the latent factors of users as recipients of tweets and the latent factors of users as publishers of tweets while the WS is responsible for computing the latent factors of words in the tweet vocabulary. By separating the latent factors of words from the latent factors of users, neither the SP nor the WS can learn the contents of users' tweets and users' interests without colluding with each other (more details in Section 5).

4.2 Threat Model

In *pTwitterRec*, we assume both the SP and the WS to be honest but curious; that is, they follow our protocol, but are curious about *passively* learning the contents of users' tweets and users' interests. More specifically, we assume that neither the SP nor the WS creates spurious users or falsifies user requests in order to learn secrets from each other. We consider this assumption to be reasonable in OSNs, because it is not in OSN providers' best interests to lose users as a result of any suspicion of malicious behaviors [12]. Furthermore, we assume that the SP and the WS do not collude.

As mentioned in the system model, we assume that a user's tweets are encrypted with a secret key shared only among the user's approved followers. No entities can decrypt the tweets without the appropriate decryption key. Both the SP and the WS are interested in learning the contents of a user's tweets. In addition, both the SP and the WS are interested in inferring users' interests by learning users' preference of specific words. For example, a user who exhibits a strong preference for words "Ford" and "Toyota" is most likely to be interested in cars.

Furthermore, we assume that the social relations between users are not hidden from the SP in *pTwitterRec*. *pTwitterRec* works for Twitter-like systems where user's social relations and the contents of users' tweets are both hidden from the provider. However, *pTwitterRec* may lose some recommendation accuracy if neither the SP nor users are able to compute some of the explicit features that depend on user's social relations (more details in Section 5.3).

5. pTwitterRec

In this section, we present the main components of *pTwitterRec*.

5.1 Design Overview

The *main challenges* are: ***a)*** when using stochastic gradient descent to learn the recommendation model, for any given tuple <*u*, *k*, *h*> in *D*, update the parameters of the model using equations 5-9 without revealing the words contained in tweets *k* and *h* to any unauthorized entities; ***b)*** when the model is learned, only user *u* is allowed to compute $p_u^T q_w$ where p_u is the latent factor of user *u* as the recipient of tweets and q_w is the latent factor of word *w*, because the result of $p_u^T q_w$ represents user *u*'s preference of word *w* and thereby potentially reveals user *u*'s interest.

We make the following *observations* in relation to equations 5-9: ***a)*** computing $\hat{e}$ in equation 9 does not require knowing the specific words contained in tweets *k* and *h*. Instead, we only need to know the result of $\frac{1}{Z^k}\sum_{w\in T_k} q_w - \frac{1}{Z^h}\sum_{w\in T_h} q_w$; ***b)*** Similarly, updating p_u using equation 5 and updating $d_{p(k)}$ using equation 7 only require knowing the result of $\frac{1}{Z^k}\sum_{w\in T_k} q_w - \frac{1}{Z^h}\sum_{w\in T_h} q_w$; ***c)*** updating b_j using equation 8 only requires knowing the result of $(r_j^{u,k} - r_j^{u,h})$ which is the difference between the explicit features computed from tweets *k* and *h* regarding user *u*, and the result of $\frac{1}{Z^k}\sum_{w\in T_k} q_w - \frac{1}{Z^h}\sum_{w\in T_h} q_w$; ***d)*** finally, updating q_w using equation 6 does not require to know the identity of user *u*. Instead, we only need to learn the result of $\hat{e}p_u$.

Intuition. Based on the above observations, we split the task of learning the parameters of the recommendation model among user *u*, the SP and the WS. User *u* is responsible for selecting tweets *k* and *h* which constitute the tuple <*u*, *k*, *h*>. User *u* and the SP cooperate to compute explicit features from tweets *k* and *h* and compute $(r_j^{u,k} - r_j^{u,h})$ without revealing the content of the tweets to the SP. The SP is responsible for updating b_j, P_u and $d_{p(k)}$ while the WS is responsible for updating q_w. The SP updates b_j, P_u and $d_{p(k)}$ upon receiving the result of $\frac{1}{Z^k}\sum_{w\in T_k} q_w - \frac{1}{Z^h}\sum_{w\in T_h} q_w$ from the WS without learning the words contained in tweets *k* and *h*. The WS updates q_w upon receiving the result of $\hat{e}p_u$ from the SP without learning the identity of user *u*. As a result, neither the SP nor the WS can calculate $p_u^T q_w$ without a coalition between them.

The main components of *pTwitterRec* include: ***a)*** word indexing, ***b)*** explicit feature computation and training sample submission, ***c)*** model learning, and ***d)*** tweet publishing, receiving and ranking. We will describe each component in details in following subsections.

Table 1. Explicit features (tweet *k* published by user *p* received by user *u*)

Feature	Category	Description	Computed By
Co-follow Score	Relation	The similarity of followee sets of between user *u* and user *p*	SP
Mention Score	Relation	The number of times user *u* has mentioned user *p* in his previous tweets	SP
Friend	Relation	1 when user *u* and user *p* follow each other and 0 otherwise	SP
Relevance to Tweet History	Content-relevance	The relevance between tweet *k* and the posting history of user *u*	User *u*
Relevance to Retweet History	Content-relevance	The relevance between tweet *k* and the retweeted history of user *u*	User *u*
Relevance to Hash Tags	Content-relevance	The count of words in tweet *k* that ever appeared as hash tags through user *u*'s posting history	User *u*
Length of Tweet	Twitter Specific	The number of words contained in tweet *k*	User *u*
Hash Tag Count	Twitter Specific	The number of hash tags contained in tweet *k*	User *u*
URL Count	Twitter Specific	The number of URLs contained in tweet *k*	User *u*
Retweet Count	Twitter Specific	The number of times tweet *k* has been retweeted	SP
Mention Count	Publishers' Authority	The times user *p* is mentioned in all tweets	SP
Followee Count	Publishers' Authority	The number of users who user *p* follows	SP
Follower Count	Publishers' Authority	The number of users who follow user *p*	SP
Tweet Count	Publishers' Authority	The number of tweets ever posted by user *p*	SP

5.2 Word Indexing

In *pTwitterRec*, user *u* is responsible for selecting tweets *k* and *h* (where tweet *k* is some tweet that user *u* has previously retweeted and tweet *h* is some tweet that user *u* has not retweeted) and preparing the tuple <*u*, *k*, *h*> in *D* before submitting it to the SP. User *u* intends to keep the content of the tweets, i.e., the words contained in tweets *k* and *h*, hidden from the SP. In addition, even though the WS only manages the latent factor of words and will not learn the identity of user *u* at the model learning stage (more details in Section 5.3), user *u* wants to keep the content of the tweets hidden from the WS as well, because the tweets may contain some words that potentially reveal the identity of user *u*. On the other hand, in order to update the latent factor of each word contained in tweets *k* and *h*, the WS must be able to uniquely index each word. Therefore, each word contained in *D* must be uniquely indexed in a manner that the WS does not learn the mapping between the word and the corresponding index.

Since all users communicate with the SP, we require the SP to be the central server responsible for indexing words using some secret that is unknown to the WS. Note that when users request the indexes of words from the SP, users want to keep the words and the corresponding indexes hidden from the SP. We propose that the SP adopts a *deterministic commutative encryption* scheme such as *Pohlig-Hellman encryption* [24], denoted as E_{comm}, to generate the unique index for each word. Loosely speaking, an encryption scheme is *commutative* if a message encrypted by key k_1 first and then by key k_2 can be decrypted by the decryption key corresponding to k_1 to reveal the message singly encrypted by k_2. Assuming that user *u* is requesting the index of word *w* from the SP, the protocol works as follows: during system setup, both the SP and user *u* generate a separate secret encryption key for the commutative encryption scheme, denoted as k_{sp} and k_u respectively. Then user *u* encrypts word *w* using E_{comm} with the secret key k_u and sends the encrypted result to the SP. Let the encrypted result be $E_{comm}(w, k_u)$. Because the SP does not know the corresponding decryption key for k_u, the SP cannot learn word *w*. Upon receiving $E_{comm}(w, k_u)$, the SP further encrypts it using E_{comm} with the SP's secret key k_{sp} and sends $E_{comm}(E_{comm}(w, k_u), k_{sp})$ back to user *u*. Finally, user *u* decrypts $E_{comm}(E_{comm}(w, k_u)$ with the corresponding decryption key for k_u. Because of the commutative property of the encryption scheme E_{comm}, user *u* obtains $E_{comm}(w, k_{sp})$ and computes $index_w = H(E_{comm}(w, k_{sp}))$ where *H* is the SHA-224 hash function and $index_w$ is the index of word *w*. Because of the deterministic property of the encryption scheme E_{comm}, the index of word *w* is unique in the whole system. In order to reduce the online computational and communication overhead on the user side, an alternative approach is to require that the SP pre-computes the indexes of popular words and users pre-download the indexes beforehand without revealing their interests to the SP. For some obscure words that are not pre-computed by the SP, users adopt the commutative encryption scheme as mentioned above to request the indexes from the SP.

5.3 Explicit Feature Computation and Training Sample Submission

For each tweet in *D*, information such as the quality of the tweet (e.g., the number of URLs contained in the tweet) and the authority of the publisher (e.g., the number of followers) can be indicated as features, which explicitly reflect the possibility of a user retweeting the tweet, known as explicit features. Chen et al. [10] propose four categories of explicit features: relation features, content-relevance features, twitter-specific features and publishers' authority features.

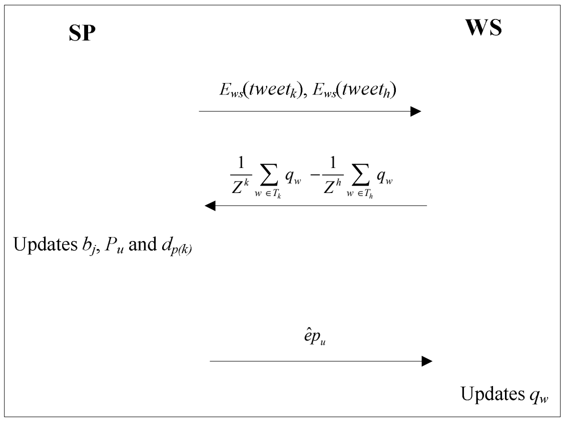
SP
WS
$E_{ws}(tweet_k)$, $E_{ws}(tweet_h)$
$\frac{1}{Z^k}\sum_{w\in T_k} q_w - \frac{1}{Z^h}\sum_{w\in T_h} q_w$
Updates b_j, P_u and $d_{p(k)}$
$\hat{e}p_u$
Updates q_w

Figure 1. Privacy-preserving model learning.

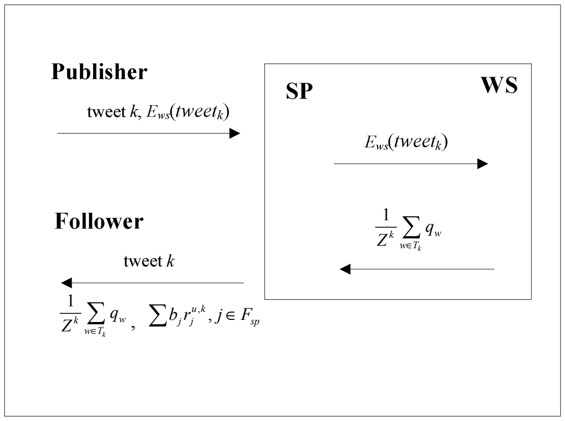
Publisher
tweet k, $E_{ws}(tweet_k)$
SP
WS
$E_{ws}(tweet_k)$
Follower
$\frac{1}{Z^k}\sum_{w\in T_k} q_w$
tweet k
$\frac{1}{Z^k}\sum_{w\in T_k} q_w$, $\sum b_j r_j^{u,k}, j\in F_{sp}$

Figure 2. Tweet publishing, receiving and ranking.

In *pTwitterRec*, we split the task of computing explicit features between users and the SP (as listed in Table 1). Users are responsible for computing those features that depend on the content of the tweet, for example, *relevance to tweet history* feature, which estimates the relevance between the tweet and the user's previous tweet history. The SP is responsible for computing those features that depend on non-sensitive global information that is available to the SP, for example, *co-follow score* feature, which estimates the similarity of the followee sets of the recipient and the publisher of the tweet.

For a given tuple <*u*, *k*, *h*> in *D*, user *u* computes the explicit features that she is responsible for computing. Let the *j*-th explicit feature computed from tweets *k* and *h* be $r_j^{u,k}$ and $r_j^{u,h}$ respectively. User *u* calculates $r_j^{u,k}$ - $r_j^{u,h}$ and attaches the result to the tuple before sending it to the SP.

Finally, user *u* cannot simply replace each word contained in tweets *k* and *h* with the corresponding index when submitting the tuple <*u*, *k*, *h*> to the SP, because the SP can conveniently learn each word by decrypting the index with its decryption key. We propose that during system setup, the WS generates a pair of public and private keys (Puk_{ws}, Prk_{ws}) for a probabilistic public-key encryption scheme denoted E_{ws}. For tweet *k* in the tuple, user *u* randomly mixes the order of the indexes of all words contained in tweet *k*, concatenates all indexes altogether, and encrypts the indexes using E_{ws} with the WS's public key Puk_{ws}. Consequently, the SP learns neither the indexes of words contained in the tweet nor the frequency of each word contained in *D* while the WS only learns the index of each word at the model learning stage. The WS does learn the frequency of each word and might be able to correlate the most popular words with their indexes at certain probabilities. However, we argue the impact of such attacks is negligible for reasons explained in Section 6.

As a result, for a given tuple <*u*, *k*, *h*> in *D*, the final data submitted to the SP by user *u* is as follows:

$$< u, \{r_j^{u,k} - r_j^{u,h} \mid j \in F_u\}, p(k), p(h), E_{ws}(tweet_k), E_{ws}(tweet_h) > \quad (10)$$

Where $E_{ws}(tweet_k)$ denotes the encrypted set of indexes of words contained in tweet *k* as previously described, F_u denotes the set of explicit features computed by user *u*, and *p*(*k*) denotes the identities of the publisher of tweets *k*. Upon receiving the data submitted by user *u*, the SP computes explicit features (as listed in Table 1) that the SP is responsible for computing from tweets *k* and *h*. Combined with the data received from user *u*, we denote the final training dataset for the given tuple <*u*, *k*, *h*> in *D* as *d*<*u*, *k*, *h*>.

5.4 Model Learning

For tweet k received by user *u*, in addition to explicit features computed as described in Section 5.3, Chen et al. [10] use a latent factor model (details in Appendix A.1) to capture user *u*'s interests in tweet *k* and propose three categories of latent features: the latent factors of user *u* as a recipient of tweet *k* in the latent feature space R^d denoted as p_u, the latent factor of the publisher of tweet *k* in R^d denoted as $d_{p(k)}$, and the latent factor of each word *w* contained in tweet *k* in R^d denoted as q_w. User *u*'s interest in tweet *k* is captured by measuring the affinity between user *u* and the words contained in tweet *k* and the affinity between user *u* and the publisher of tweet *k* in the latent feature space R^d. At the model learning stage, the SP and the WS cooperate to learn the values of these latent factors, in addition to the weight vector for explicit features by performing stochastic gradient descent as described in Section 3.3.

In *pTwitterRec*, during system setup, the SP initializes the weight b_j for all explicit features, the latent factor p_u of each user as the recipient of tweets and the latent factor $d_{p(k)}$ of each user as the publisher of tweets with random values. For each training dataset *d*<*u*, *k*, *h*> corresponding to <*u*, *k*, *h*> in *D*, the SP and the WS engage in the following protocols (as depicted in Figure 1):

a) The SP forwards the encrypted set of word indexes (denoted as $E_{ws}(tweet_k)$ and $E_{ws}(tweet_h)$) contained in *d*<*u*, *k*, *h*> to the WS. The WS obtains the corresponding word indexes by decrypting with its private key Prk_{ws}. If the WS has never come across the index of word *w* before, the WS initializes the latent factor for word *w* with random values, denoted as q_w. Then, the WS calculates $\frac{1}{Z^k}\sum_{w\in T_k} q_w - \frac{1}{Z^h}\sum_{w\in T_h} q_w$ and sends the result back to the SP. Note that the WS does not know the actual words but only learns the index of each word contained in tweets *k* and *h*.

b) Upon receiving $\frac{1}{Z^k}\sum_{w\in T_k} q_w - \frac{1}{Z^h}\sum_{w\in T_h} q_w$ from the WS, the SP calculates $\hat{e}$ using equation 9 and updates parameters b_j, p_u and $d_{p(k)}$ using equations 5, 7 and 8.

c) The SP computes $\hat{e}p_u$ and sends the result to the WS. The WS updates q_w for each word *w* contained in tweets *k* and *h* using equation 6. Note that the WS does not learn the identity of user u.

The SP and the WS loop over all training datasets in D and update the parameters of the tweet recommendation model in the same manner as mentioned above. At the end of the model learning stage, the SP learns b_j for each explicit feature, p_u for users as recipients of tweets and $d_{p(k)}$ for users as publishers of tweets while the WS learns q_w for words in the tweet vocabulary.

5.5 Tweet Publishing, Receiving and Ranking

Once the tweet recommendation model is learned, it can be used to predict users' interests in tweets and rank the tweets accordingly (as depicted in Figure 2).

In *pTwitterRec*, users are primarily responsible for ranking tweets with the help of the SP and the WS. When the recommendation model is learned, user u requests from the SP her personal latent factor p_u, the latent factors of all her followees as publishers, denoted as d_f where user f is a followee of user u, and the system-wide weight b_j for explicit features. Note that user u only needs to retrieve these parameters once.

When publishing a tweet, the user looks up the indexes of words contained in the tweet to be published, mixes the order of these indexes randomly and concatenates them together, encrypts the indexes using E_{ws} with the WS's public key Puk_{ws} and attaches the encrypted indexes to the tweet before sending it to the SP. Similarly to model learning, the SP forwards the encrypted indexes to the WS, which calculates $\frac{1}{Z^k}\sum_{w \in T_k} q_w$ for words contained in the tweet and sends the result back to the SP. The SP computes the sum of explicit features (denoted as $\sum b_j r_j^{u,k}, j \in F_{sp}$ where F_{sp} denotes the set of explicit features that the SP is responsible for computing), attaches the sum along with $\frac{1}{Z^k}\sum_{w \in T_k} q_w$ which was previously received from the WS to the tweet before distributing it to corresponding followers.

Upon receiving the tweet and the attached values, user u decrypts the tweet, calculates the explicit features that the user is responsible for computing, and computes the predicted rating for the tweet using equation 4. Tweet ranking works as follows: When the actual user, not the *pTwitterRec* client software, wants to read her tweets, this process is done for all tweets that have arrived since her last update and then the tweets are shown to her in ranked order.

6. DISCUSSION

In this section, we discuss the security and privacy aspects of *pTwitterRec* and a possible extension.

6.1 Security Analysis

We discuss possible attacks against *pTwitterRec*.

Frequency analysis attacks. At the model learning stage, the WS decrypts the encrypted indexes of the words contained in the tweets submitted by users as described in Section 5.4. Therefore, the WS learns the frequency of each word that has appeared in all tweets contained in the training datasets D while not knowing the actual words. Should the WS have the background knowledge of the popularity of each word in the tweet vocabulary, the WS might be able to correlate the most popular words with their indexes at certain probabilities. However, we argue that those words that are most vulnerable against such frequency analysis attacks are exactly those words that are most common among all users and therefore reveal little personal information regarding an individual user. In regards of the SP, even though the SP knows the identities of the users included in D, the SP cannot learn the frequencies of words because the indexes were encrypted using probabilistic encryption scheme E_{ws} with the WS's public key Puk_{ws}.

Collusion attacks. In our threat model, we assume that the SP and the WS do not collude. In addition, since the SP and the WS are honest-but-curious adversaries, they would not create phantom users to interact with other parties. The SP might try to collude with some legitimate users. A collusion between the SP and a user does not reveal the indexes of words contained in other users' tweets as they are encrypted using probabilistic encryption scheme E_{ws} (we require that E_{ws} is also secure against chosen-plaintext attacks.) with the WS's public key Puk_{ws}. However, the SP can learn the latent factors of some words through the colluding user and thereby infer other users' interests. Similarly, a collusion between the WS and a user discloses the mapping of words and corresponding indexes. However, we claim that users who collude with either the SP or the WS lose some of their own privacy. A collusion among a group of users does not pose threats to other users, because in *pTwitterRec* users are only allowed to obtain their own personal latent factors from the SP, as described in Section 5.5.

Poisoning attacks. Some malicious users may inject false training datasets such as falsified explicit features to render the tweet recommendation model less effective. Prior work such as Orca [2] has been proposed to detect such poisoning attacks. Poisoning attacks are not introduced as a result of adopting *pTwitterRec*. We leave the full investigation of the impact of such attacks on the recommendation accuracy and the applicability of existing defense mechanisms to future work.

6.2 Privacy Analysis

In this subsection, we examine the user information disclosed to the SP and the WS at each stage of *pTwitterRec* and analyze the privacy threats.

Word indexing. When user u requests the index of word w from the SP, the SP learns neither word w nor the corresponding index, because word w is encrypted using commutative encryption with a secret key known only to user u. Upon receiving the index of word w from the SP, user u caches the index corresponding to word w locally so user u does not need to request the index of the same word again in the future. Therefore, the SP learns no information other than that user u has made such a request. If the SP pre-computes the indexes for all popular words in the tweet vocabulary as described in Section 5.2, user u reveals no personal information by downloading such indexes from the SP in advance.

Explicit feature computation and training sample submission. For a given tuple <u, k, h> in D, when user u finishes calculating the explicit features for tweets h and k and submitting the corresponding training dataset to the SP as described in Section 5.3, the SP learns: ***a)*** the identities of the publishers of tweets k and h, which is not private information in our threat model; ***b)*** the encrypted indexes for words contained in tweets k and h, which are encrypted using E_{ws} with the WS's public key Puk_{ws}. The SP learns neither the words contained in the tweets nor their indexes. In addition, the SP does not even learn how frequent the same word has appeared among all tweets in D because E_{ws} is a probabilistic encryption scheme; ***c)*** the difference between the explicit features (computed by user u) computed from tweets k and h, denoted as $\{r_j^{u,k} - r_j^{u,h} \mid j \in F_u\}$. Among all explicit features computed by user u, features *Relevance to Tweet History*, *Relevance to Retweet History* and *Relevance to Hash Tags* do not leak any personal information about user u because all tweets received and published by user u are hidden from the SP. Features *Length of Tweet*, *Hash Tag Count* and *URL Count* do reveal the number of words, the number of hash tags and the number of URLs contained in a tweet. However, we do not consider such information to be sensitive for user u. In addition, user u only submits the difference between those features computed from tweets k and h; ***d)*** the SP computes some explicit features of tweets k and h as listed in Table 1. However, those features only depend on non-sensitive information, which is already available to the SP, such as user u's social relations, and therefore do not leak any private information about user u.

Model learning. At the model learning stage, the SP and the WS cooperate to update the parameters of the tweet recommendation model. For a given tuple <u, k, h> in D, ***a)*** at the first step of the protocol, the SP only forwards the encrypted indexes for words contained in tweets k and h to the WS. The WS does not learn the original words contained in tweets k and h but only learns their indexes. In addition, the WS learns neither the identity of user u nor the identity of the publishers of tweets k and h. Furthermore, we require that users randomly mix the order of indexes of words contained in the tweet before encrypting and submitting them, so the WS cannot identify words by analyzing the pattern of the order of words appearing in tweets. The WS does learn the frequency of each word that has appeared in all tweets contained in D while not knowing the actual words. However, as analyzed in Section 6.1, such frequency analysis attacks reveal little personal information regarding each individual user. At the end of the first step, the SP learns the result of $\frac{1}{Z^k}\sum_{w \in T_k} q_w - \frac{1}{Z^h}\sum_{w \in T_h} q_w$ from the WS, which does not reveal the individual latent factor of each word because all word indexes were encrypted using probabilistic encryption scheme E_{ws} with the WS's public key Puk_{ws}; ***b)*** at the third step of the protocol, the WS learns the result of $\hat{e}p_u$ from the SP, which does not reveal the identity of user u; ***c)*** at the end of the model learning stage, the SP learns the latent factor p_u of user u as recipients of tweets and the latent factor $d_{p(k)}$ of user $p(k)$ as the publisher of tweet k (note that p_u is a vector in the latent feature space R^d and p_u alone does not disclose any personal information of user u). The SP is able to learn the closeness of the social relation between users u and $p(k)$ by computing $p_u^T d_{p(k)}$. However, in *pTwitterRec*, we assume that the social relations between users are not hidden from the SP, as described in the threat model; ***d)*** the SP cannot replace the WS by updating the latent factors of words as well, because the indexes for words are encrypted by user u using the probabilistic encryption scheme with the WS's public key; ***e)*** the WS cannot replace the SP, because the WS cannot compute the model all by itself without knowing the explicit features, the identity of the recipient of the tweet, and the identity of the publisher of the tweet, which are only known to the SP.

Tweet publishing, receiving and ranking. For tweet k received by user u, the SP and the WS both only learn the value of $\frac{1}{Z^k}\sum_{w \in T_k} q_w$ for all words contained in tweet k but do not know the words contained in tweet k. In addition, neither the SP nor the WS can infer user u's interest by computing $p_u^T q_w$ for any word w (which represents user u's preference over word w) without colluding with each other. Furthermore, for user u, we require that user u is only allowed to retrieve her own personal latent factor p_u from the SP and therefore user u cannot learn other users' interests.

6.3 Extension

In *pTwitterRec*, the SP is not only responsible for managing the social relations between users and distributing users' tweets to their followers as an OSN provider, but also responsible for recommending personalized interesting tweets to users (in cooperation with the WS and users) as a recommendation service provider. However, *pTwitterRec* can be conveniently adapted to support a third-party privacy-preserving tweet recommendation service that is independent of the OSN provider. The benefit of implementing *pTwitterRec* as a third-party service is that there would be no requirements of any changes to the existing OSN provider while still providing users with personalized tweet recommendations (and hiding the contents of users' tweets and user's interests from the tweet recommendation service provider).

With the new *pTwitterRec* client application, users receive/publish tweets from/to the OSN provider, and interact with the third-party tweet recommendation service provider (consisting of a separate SP and WS) to learn the recommendation model without leaking any sensitive information to the recommendation service provider. Upon the completion of the model learning stage, users are able to rank tweets received from the OSN provider. *pTwitterRec* may lose some recommendation accuracy because the recommendation service provider is unable to compute some explicit features such as features that depend on users' social relations. However, users may be able to compute such features if they know their followees' social relations.

7. IMPLEMENTATION

To demonstrate the practicality of our framework, we implemented a prototype that simulates the model learning stage of *pTwitterRec*, which is the core component of our framework. The prototype consists of: a server acting as the SP that takes the training datasets D as input and updates the weight b_j for each explicit feature, the latent factor p_u of user u as the recipient of the tweets, and the latent factors $d_{p(k)}$ and $d_{p(h)}$ of users $p(k)$ and $p(h)$ as publishers of tweets k and h for every tuple <u, k, h> in D; another server acting as the WS that communicates with the SP as described in Section 5.4 and updates the latent factor q_w of each word w contained in D.

Our prototype uses the *SVDFeature* toolkit [37] to implement gradient stochastic descent, which is used for updating abovementioned parameters at the model learning stage. We choose *SVDFeature* since it is open source and well-documented. *SVDFeature* is a toolkit designed to efficiently solve large-scale collaborative filtering problems with auxiliary information by performing gradient stochastic descent. Unlike traditional approaches that require writing a specific solver for each recommendation model, *SVDFeature* provides a general solution for collaborative filtering problems and allows developing new recommendation models by defining new features.

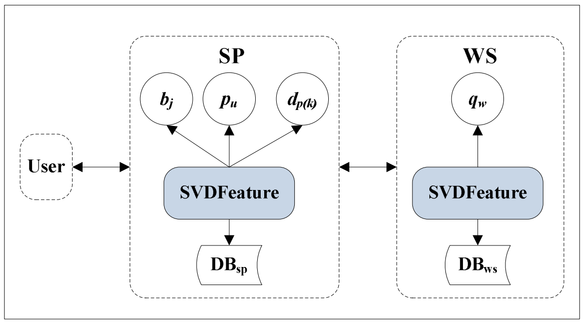

Figure 3. Server-side architecture.

In our prototype, we define the explicit features described in Section 5.3 as global features in *SVDFeature*, the latent factors of users (as recipients of tweets) as user features in *SVDFeature*, and the latent factors of words contained in the tweet vocabulary in addition to the latent factors of users (as publishers of tweets) as item features in *SVDFeature*. Figure 3 depicts the architecture of our prototype. The SP and the WS each maintains a separate database. At the end of the model learning stage, the SP stores in its database the final values of the system-wide weight b_j for each explicit feature, the latent factor p_u of each user as the recipient of tweets, and the latent factors $d_{p(k)}$ of each user as the publisher of tweets; the WS stores in its database the latent factor q_w of each word in the tweet vocabulary. We leave the implementation of the complete framework to future work.

8. EVALUATION

In this section, we analyze and evaluate the overhead particularly on the user side due to the introduction of our privacy protection framework *pTwitterRec*, compared with the original tweet recommendation algorithm [10] with no user privacy protection. Our framework only adds privacy protection to the tweet recommendation algorithm without modifying the original recommendation model [10]. Thus, there is no loss of recommendation accuracy (please refer to [10] for the thorough evaluation of recommendation accuracy).

We assume that users run the *pTwitterRec* client application on a smartphone with limited computation power and memory. In our evaluation, we use a Google Nexus Four phone featuring a 1.512 GHz quad-core Krait CPU and 2 GB of RAM. On the other hand, we assume that both the SP and the WS have reasonably unlimited computation power and storage space. We use an 8 core server featuring Intel Xeon 2.40 GHz CPU with 8 GB RAM as the SP/WS. We assume that there is a fast and persistent network connection between the SP and the WS and the communication overhead between them is negligible. We analyze the overhead at each stage of our framework in following subsections.

8.1 Word Indexing

As described in Section 5.2, assuming that user u does not know the index of word w, user u first encrypts word w using commutative encryption E_{comm} with her secret key k_u and then sends the encrypted word to the SP. The SP further encrypts the received encrypted word using E_{comm} with its secret key k_{sp} and returns the result to user u. Finally, user u obtains the index of word w by decrypting the result received from the SP with her decryption key and then computing the hash value of the result using SHA-224. Therefore, to obtain the index of a new word, it requires user u to perform one encryption and one decryption using commutative

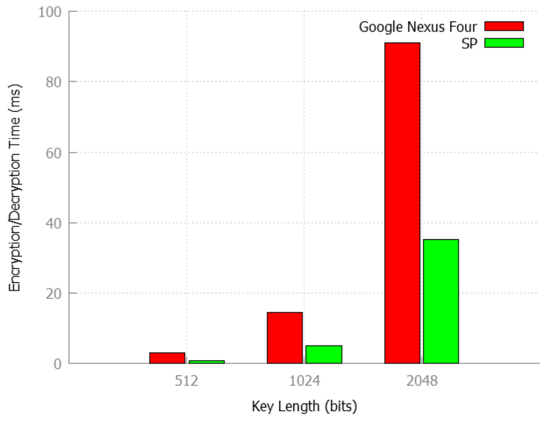

Figure 4. Pohlig-Hellman encryption/decryption.

encryption E_{comm} and perform one hash function while it requires the SP to perform one encryption using E_{comm}. The computational cost for carrying out one hash function is negligible and therefore we only evaluate the computational cost for commutative encryption/decryption. We adopt *Pohlig-Hellman encryption* (described in Appendix A.2) as E_{comm} for our evaluation because of its deterministic commutative property (note that any encryption scheme that is both *deterministic* and *commutative* can be used as E_{comm} in our protocol, as described in Section 5.2). We measure the execution time of a single *Pohlig-Hellman* encryption/decryption on the smartphone and on the server. We vary the key length, carry out 10,000 runs for each key length and average the results. Figure 4 shows the results. With a 2048-bit key, it takes approximately 91.1 ms for the smartphone and 35.2 ms for the SP to carry out one *Pohlig-Hellman* encryption/decryption. The communication overhead for user u is to send and receive one encrypted word. Therefore, with a 2048-bit key size, the communication overhead for user u is approximately 512 bytes to obtain the index of one word. It is reported that there are approximately 15 words [36] in each tweet on average. Assuming that each user submits n (n = 392 in [10]) tweets to the SP as training datasets, the total communication overhead for a user to obtain all indexes from the SP is less than 7.5n KB and the total computational overhead is less than 2.7n seconds. Note that user u caches the index of word w locally and thereby only needs to request the index of word w from the SP once even if word w appears in multiple tweets. In addition, to further reduce the overhead on the user side, as described in Section 5.2, we propose that the SP precomputes the indexes for popular words and users pre-download the indexes for these popular words beforehand without revealing their interests to the SP.

8.2 Explicit Feature Computation and Training Sample Submission

In *pTwitterRec*, users are responsible for computing six of the explicit features for each tweet contained in the training samples. Compared with the cryptographic overhead evaluated in Section 8.1, the computational overhead incurred by explicit feature computation is negligible for users. For each tweet contained in the training samples, *Relevance to Tweet History* feature and *Relevance to Retweet History* feature each takes four bytes, and the remaining four integer-value features each takes two bytes.

Furthermore, users encrypt the indexes of words contained in the tweet using probabilistic encryption E_{ws} with the WS's public key Puk_{ws}. The size of each word index is 28 bytes (recall that users

Table 2. User side overhead (each user submits *n* tweets for training)

	Computational Overhead	Communication Overhead
Word Indexing	<2.7*n* seconds	<7.5*n* KB
Explicit Feature Computation and Training Sample Submission	0.1*n* seconds	0.7*n* KB
Model Learning	none	none
Tweet Publishing and Receiving (one tweet)	Publisher: 91.1 ms Receiver: none	Publisher: 0.7 KB Receiver: 8 bytes
Tweet Ranking	negligible	none

adopt SHA-224 to compute the indexes for words) and therefore the plaintext of all indexes for a tweet is 420 bytes on average. We propose that users encrypt all indexes using *Advanced Encryption Standard* (AES) in CBC mode with a random secret key k_r and then encrypt the secret key using RSA with the WS's public key Puk_{ws} (note that any *probabilistic* public-key encryption scheme can be used as E_{ws} in our protocol, as described in Section 5.3). The size of the ciphertext is approximately equal to the size of (all word indexes + one AES block + initialization vector + the ciphertext of the secret AES key k_r encrypted using RSA), which is about 0.7 KB in total with a 2048-bit RSA key. The computational overhead for encrypting word indexes is dominated by encrypting k_r using RSA. With the same key size, the computational cost for RSA encryption/decryption is the same as Pohlig-Hellman encryption/decryption (see Figure 4) because the compute-intensive part for both encryption schemes is one modular exponentiation. Therefore, the computational overhead at this stage for the user to submit *n* tweets is approximately 0.1*n* seconds and the communication overhead is approximately 0.7*n* KB for a user.

8.3 Model Learning

Once the SP has received all training samples from users, the SP only interacts with the WS at the model learning stage. Therefore, there is no overhead on the user side. Compared with the original recommendation algorithm [10], we split the model learning task between the SP and the WS. As we assume that the communication overhead between the SP and the WS is negligible, the only overhead incurred by *pTwitterRec* is the cost for the WS to decrypt and obtain the indexes for words contained in tweets, so that the WS can update the corresponding latent factors of these words.

8.4 Tweet Publishing, Receiving and Ranking

When publishing a tweet, the publisher carries out one probabilistic encryption E_{ws} to encrypt the indexes of all words contained in the tweet in the same manner as described in Section 8.2 and attaches the result to the tweet before publishing it. Therefore, to publish a tweet, the computational overhead for the publisher is approximately 91.1 ms and the communication overhead is approximately 0.7 KB on average. The overhead incurred on the server side is that the WS needs to perform one decryption to obtain the word indexes for each tweet. As described in Section 5.5, the SP attaches the results of $\sum b_j r_j^{u,k}$ and $\frac{1}{Z^k}\sum_{w\in T_k} q_w$ to the tweet before forwarding it to the followers. Therefore, the communication overhead for the followers to receive a tweet is approximately the size of two floating point number, which is eight bytes. We do not consider here the overhead incurred by decrypting tweets using the publisher's secret key that is shared only among the publisher's followers, which is out of the scope of our work.

For tweet ranking, upon receiving a tweet from the SP, the user only needs to compute the explicit features and then predicts the rating for the tweet using equation 4. Therefore, the overhead incurred by tweet ranking on the user side is negligible and there is no communication overhead.

8.5 Overall Overhead

Users may run the client application on a smartphone with limited computation power and memory. Therefore, we illustrate the feasibility of *pTwitterRec* by analyzing the overall overhead incurred on the user side. We summarize the results in Table 2.

For model learning, assuming that each user submits *n* tweets to the SP for training, the computational overhead for a user is approximately 2.7*n* seconds and the communication overhead is approximately 7.5*n* KB in total assuming that the user requests the index of every single word contained in the tweets. Note that a user only needs to submit training samples once and the tweet recommendation model can be learned offline between the SP and the WS. Therefore, there are no strict real-time requirements for a user to submit training samples and it can take place when the user's phone is not busy and when it has a Wi-Fi connection to the Internet.

To publish a tweet, the computational overhead for the publisher is 91.1 ms and the communication overhead is 0.7 KB. The publisher can choose to not participate in tweet recommendations by just publishing tweets without attaching the encrypted word indexes. However, the publisher's followers may overlook her tweets as the followers cannot predict the ratings for her tweets and rank her tweets. For every received tweet, there is no computational cost for the receiver and the communication overhead for the receiver is eight bytes per tweet.

For tweet ranking, the computational overhead for the user is negligible and there is no communication overhead.

Summary. The user side overhead incurred by *pTwitterRec* for tweet recommendation model learning is reasonable and the model learning between the SP and the WS can take place offline. The overhead for publishing and receiving tweets incurred by *pTwitterRec* is small and the overhead for ranking tweets is negligible.

9. CONCLUSIONS

In this paper, we present *pTwitterRec*, the first privacy-preserving personalized tweet recommendation framework that provides users with the benefits of tweet recommendations while keeping the content of tweets and users' interests hidden from other unauthorized entities including the provider. We introduce a semi-trusted third server (WS) to compute the tweet recommendation model, in cooperation with users and the SP. The implementation and evaluation show that *pTwitterRec* is practical and only introduces reasonable overhead. Our future work includes implementing the complete framework of *pTwitterRec* and evaluating the performance in the real world.

10. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. This work is supported by a Google Focused Research Award, the Ontario Research Fund, and the Natural Sciences and Engineering Research Council of Canada.

11. REFERENCES

[1] Aïmeur, E., Brassard, G., Fernandez, J.M. and Mani Onana, F.S. Alambic: a privacy-preserving recommender system for electronic commerce. *International Journal of Information Security*. 7, 5 (2008), 307–334.

[2] Bay, S.D. and Schwabacher, M. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2003), 29–38.

[3] Beato, F., Kohlweiss, M. and Wouters, K. Scramble! your social network data. *Proceedings of the 11th International Conference on Privacy Enhancing Technologies* (2011), 211–225.

[4] Billsus, D. and Pazzani, M.J. Learning collaborative information filters. *Proceedings of the 15th International Conference on Machine Learning* (1998), 46–54.

[5] Blei, D.M., Ng, A.Y. and Jordan, M.I. Latent dirichlet allocation. *Journal of Machine Learning research*. 3, (2003), 993–1022.

[6] Breese, J.S., Heckerman, D. and Kadie, C. Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the 14th conference on Uncertainty in Artificial Intelligence* (San Francisco, CA, USA, 1998), 43–52.

[7] Canny, J. Collaborative filtering with privacy. *Proceedings of 2002 IEEE Symposium on Security and Privacy* (2002), 45–57.

[8] Canny, J. Collaborative filtering with privacy via factor analysis. *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2002), 238–245.

[9] Chen, J., Nairn, R., Nelson, L., Bernstein, M. and Chi, E. 2010. Short and tweet: experiments on recommending content from information streams. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), 1185–1194.

[10] Chen, K., Chen, T., Zheng, G., Jin, O., Yao, E. and Yu, Y. Collaborative personalized tweet recommendation. *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2012), 661–670.

[11] Conti, M., Hasani, A. and Crispo, B. Virtual private social networks. *Proceedings of the 1st ACM Conference on Data and Application Security and Privacy* (2011), 39–50.

[12] De Cristofaro, E., Soriente, C., Tsudik, G. and Williams, A. Hummingbird: privacy at the time of Twitter. *Proceedings of 2012 IEEE Symposium on Security and Privacy (SP)* (May 2012), 285 –299.

[13] Diaz-Aviles, E., Drumond, L., Schmidt-Thieme, L. and Nejdl, W. Real-time top-n recommendation in social streams. *Proceedings of the 6th ACM Conference on Recommender Systems* (2012), 59–66.

[14] Duan, Y., Jiang, L., Qin, T., Zhou, M. and Shum, H.-Y. An empirical study on learning to rank of tweets. *Proceedings of the 23rd International Conference on Computational Linguistics* (2010), 295–303.

[15] Feldman, A.J., Blankstein, A., Freedman, M.J. and Felten, E.W. Social networking with Frientegrity: privacy and integrity with an untrusted provider. *Proceedings of the 21st USENIX Conference on Security Symposium* (Berkeley, CA, USA, 2012), 31–31.

[16] Guha, S., Tang, K. and Francis, P. NOYB: Privacy in online social networks. *Proceedings of the 1st Workshop on Online Social Networks* (2008), 49–54.

[17] Hannon, J., Bennett, M. and Smyth, B. Recommending twitter users to follow using content and collaborative filtering approaches. *Proceedings of the 4th ACM Conference on Recommender Systems* (2010), 199–206.

[18] Hong, L., Bekkerman, R., Adler, J. and Davison, B.D. Learning to rank social update streams. *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2012), 651–660.

[19] Joachims, T. Optimizing search engines using clickthrough data. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2002), 133–142.

[20] Koren, Y., Bell, R. and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*. 42, 8 (2009), 30–37.

[21] Kunegis, N.N.T.G.J. and Alhadi, A.C. Bad news travel fast: a content-based analysis of interestingness on twitter. *Proceedings of ACM Web Science Conference* (2011).

[22] Lucas, M.M. and Borisov, N. Flybynight: mitigating the privacy risks of social networking. *Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society* (2008), 1–8.

[23] Luo, W., Xie, Q. and Hengartner, U. Facecloak: an architecture for user privacy on social networking sites. *Proceedings of International Conference on Computational Science and Engineering* (2009), 26–33.

[24] Pohlig, S. and Hellman, M. An improved algorithm for computing logarithms over GF (p) and its cryptographic significance. *Information Theory, IEEE Transactions on*. 24, 1 (1978), 106–110.

[25] Polat, H. and Du, W. Achieving private recommendations using randomized response techniques. *Advances in Knowledge Discovery and Data Mining*. (2006), 637–646.

[26] Polat, H. and Du, W. Privacy-preserving collaborative filtering. *International Journal of Electronic Commerce*. 9, 4 (2003), 9–35.

[27] Polat, H. and Du, W. Privacy-preserving collaborative filtering using randomized perturbation techniques. *Proceedings of the 3rd IEEE International Conference on Data Mining* (Washington, DC, USA, 2003), 625–628.

[28] Polat, H. and Du, W. SVD-based collaborative filtering with privacy. *Proceedings of the 2005 ACM symposium on Applied Computing* (New York, NY, USA, 2005), 791–795.

[29] Qu, Z. and Liu, Y. Interactive group suggesting for Twitter. *Proceedings of the 49th Annual Meeting of the Association*

for Computational Linguistics: Human Language Technologies (Short Papers) (2011), 519–523.

[30] Rainie, L., Kiesler, S., Kang, R. and Madden, M. 20130905. Anonymity, privacy, and security online. *Pew Internet & American Life Project*. *http://www.pewinternet.org/PPF/r/213/report_display.asp*,

[31] Ramage, D., Dumais, S.T. and Liebling, D.J. Characterizing microblogs with topic models. *Proceedings of the 4th International AAAI Conference on Weblogs and Social Media* (2010).

[32] Reeves, S. Internet is double-edged sword in Arab revolts. *http://middle-east-online.com/english/?id=46109*

[33] Rennie, J.D. and Srebro, N. Fast maximum margin matrix factorization for collaborative prediction. *Proceedings of the 22nd International Conference on Machine Learning* (2005), 713–719.

[34] Singh, I., Butkiewicz, M., Madhyastha, H.V., Krishnamurthy, S.V. and Addepalli, S. Twitsper: tweeting privately. *IEEE Security Privacy*. 11, 3 (2013), 46–50.

[35] Yang, S.-H., Long, B., Smola, A., Sadagopan, N., Zheng, Z. and Zha, H. Like like alike: joint friendship and interest propagation in social networks. *Proceedings of the 20th International Conference on World Wide Web* (2011), 537–546.

[36] RT this: OUP Dictionary team monitors Twitterer's tweets. *http://blog.oup.com/2009/06/oxford-twitter/*

[37] SVDFeature toolkit. *http://svdfeature.apexlab.org/wiki/Main_Page*

[38] Twitter statistics. *http://www.statisticbrain.com/twitter-statistics/*

A: Appendix

A.1 Latent Factor Model

Collaborative recommendation techniques based on latent factor models have been proved to be effective in improving recommendation accuracy [20, 33]. In a basic latent factor model, users and items are mapped to a joint low-dimensional latent factor space R^d. Let p_u be the low dimensional representation of user u in the latent feature space R^d and q_i be the low dimensional representation of item i in R^d. The predicted rating of user u for item i is computed as the affinity between user u and item i in the latent feature space, i.e., the inner product of p_u and q_i in R^d.

For tweet recommendations, directly applying the basic latent factor model faces the problem of data sparsity due to the lack of retweet data. Chen et al. [9] propose decomposing the latent factor of a tweet (i.e., an item in the basic latent model) into a combination of the latent factors of words contained in the tweet. Therefore, for a given tweet k, $p_u^T \sum_{w \in T_k} q_w$ captures the impact of the words contained in tweet k on user u's rating of the tweet, where T_k is the word set contained in tweet k and q_w represents the low dimensional representation of word w in the latent feature space R^d. Furthermore, Chen et al. propose representing the publisher of tweet k as a latent factor (denoted as $d_{p(k)}$) in R^d as well and measuring the possibility of user u retweeting tweet k by considering the affinity of user i and the publisher of tweet k in the latent feature space. Therefore, $p_u^T d_{p(k)}$ captures the impact of the social relation between user u and the publisher of tweet k on user u's rating of the tweet. The values of p_u, q_w and $d_{p(k)}$ are learned through the training as described in Section 5.4.

A.2 Pohlig-Hellman Encryption

The Pohlig-Hellman encryption scheme [24] is similar to RSA. Different keys are used for encryption and decryption. However, it is not a public-key scheme, because the keys are easily derivable from each other; both the encryption and decryption keys must be kept secret.

Given a large prime p with no small factors of p - 1, each party chooses a random (e, d) pair such that $e \times d = 1 \pmod{p-1}$ where the encryption key is e and the decryption key is d. For a given message M, the encryption of M is $M^e \pmod{p}$ and for a given ciphertext C, the decryption of C is $C^d \pmod{p}$. It is straightforward to prove that the Pohlig-Hellman encryption scheme is both deterministic and commutative.

Pohlig-Hellman leaks the information whether the plaintext message M is quadratic residue (mod p) or not. In our framework, the SP computes the index of each word by encrypting the word using Pohlig-Hellman. Because the *SP* is honest but curious in the threat model, the *SP* would not manipulate the distribution of word indexes at the risk of being caught. Although we use Pohlig-Hellman encryption for examples in this paper, any encryption scheme that is both *deterministic* and *commutative* can be used in our protocol.