Privacy Preserving Collaborative Filtering from Asymmetric Randomized Encoding

Yongjun Zhao and Sherman S.M. Chow*

Department of Information Engineering
The Chinese University of Hong Kong, Hong Kong
{zy113, sherman}@ie.cuhk.edu.hk

Abstract. Collaborative filtering is a famous technique in recommendation systems. Yet, it requires the users to reveal their preferences, which has undesirable privacy implications. Over the years, researchers have proposed many privacy-preserving collaborative filtering (PPCF) systems using very different techniques for different settings, ranging from adding noise to the data with centralized filtering to performing secure multi-party computation. However, either privacy protection is unsatisfactory or the computation is prohibitively expensive.
In this work, we propose a decentralized PPCF system, which enables a group of users holding (cryptographically low-entropy) profile to identify other similar users in a privacy preserving yet very efficient way, without the help of any central server. Its core component is a novel primitive which we named as *asymmetric randomized encoding* (ARE). Similar to the spirt of other cryptographic primitives, it is asymmetric in the sense that, honest party could enjoy performance boost via precomputation with the knowledge of a profile, whilst adversary aiming to recover the hidden profile can only launch dictionary attack against each encoded profile. Thanks to the simple design of ARE, our solution is very efficient, which is demonstrated by our performance evaluation. Besides PPCF, we believe that ARE will find further applications which require a balance between privacy and efficiency.

Keywords: asymmetric randomized encoding, privacy-preserving collaborative filtering, recommendation system, peer-to-peer network

1 Introduction

Collaborative filtering [42] (CF) is a widely used data mining technique in recommendation systems. People can obtain highly personalized and accurate recommendations for books, movies, *etc.* based on their past consumption activities (or *user-profile* in the rest of the paper), such as rating a movie or buying certain commodity. With the widespread of different online communities, and people's

* Sherman Chow is supported by the Early Career Scheme and the Early Career Award of the Research Grants Council, Hong Kong SAR (CUHK 439713), and Direct Grant (4055018) of the Chinese University of Hong Kong.

willingness to share experiences and opinions, CF is getting more popular. It also becomes increasingly important in our daily life as it brings better user-experience to customers and larger revenue to service providers.

However, the win-win benefits brought by CF comes at the price of risking user privacy in various ways. For example, service providers might have the incentive to secretly sell user-profile to other parties, or they might unintentionally leak such information to public. The latter case actually happened, when Netflix released a dataset containing about 500,000 anonymous users' movie rating profile for more than 17,000 movies in an open competition for the best CF algorithm [43]. About two years later, Narayanan *et al.* [31] broke the anonymization of Netflix database by leveraging some limited auxiliary information of the users.

In the light of privacy breach [31], *privacy-preserving collaborative filtering* (PPCF) is moving towards untrusted server setting [32–34] or decentralized setting [4, 8, 9, 29, 30, 37], to eliminate the trust assumption on a centralized server. Unfortunately, many randomization techniques [32–34] have been shown to be insufficient to preserve privacy, and existing secure schemes either rely on heavy cryptographic tools or rely on additional network middleware (see Section 2 for a detailed discussion). It is fair to say designing a practical PPCF system without additional infrastructure remains an open problem.

In this paper, we tackle this challenge by formulating and proposing a novel primitive for the core functionality required in PPCF that only uses relatively lightweight cryptographic primitives. Most CF systems make use of user-to-user similarity to provide recommendations. If users are able to identify other similar users just by themselves through a peer-to-peer (P2P) network, they can then generate useful recommendation themselves. The key insight of our design philosophy is that, an honest user comes with a user profile to search for similar users, while an adversary may not be motivated to just target a specific user. We thus put our attention to devise an encoding mechanism for the user profiles, such that honest users can efficiently identify similar users, while the best an adversary can do is to launch a dictionary attack per each participated user.

1.1 Our Contributions

Firstly, we design a specialized cryptographic tool called *asymmetric randomized encoding* (ARE) that enables highly efficient privacy preserving filtering. An ARE scheme $\mathsf{ARE} = (\mathcal{P}, \mathcal{E}, \mathcal{T})$ is a tuple of three polynomial time algorithms, where $\mathcal{P}$ is the parameter generation algorithm, $\mathcal{E}$ is the encoding algorithm, and $\mathcal{T}$ is the test algorithm. Encoding algorithm $\mathcal{E}$ takes a binary string m and public parameter P as input and outputs a succinct representation $\mathcal{E}(P, m)$ of m that only leaks enough information for efficient filtering using the test algorithm $\mathcal{T}$. The "asymmetric" nature of ARE captures the property that, any honest user could efficiently run $\mathcal{T}$ algorithm with the knowledge of m, whilst adversaries without m could not due to the asymmetry in their goals and knowledge. Its "randomized" nature provides better security than any deterministic schemes. We define appropriate security notions for ARE, propose a very efficient realization, and prove its security in the random oracle model.

While our proposed construction is very simple, we view the major novelty of our work is the identifications of 1) what can we rely on for security in a decentralized setting (not even a public-key infrastructure), where everyone could be an honest user or an adversary, and 2) the core functionality required in supporting PPCF.

We then show how to easily combine this "exact filtering" tool and *locality sensitive hashing* (LSH) to support "similar user filtering", which ultimately enables a very simple PPCF protocol: after a user has identified a few "similar" users securely, she simply exchanges actual user profile in a secure channel, and then runs a collaborative filtering algorithm locally to generate recommendations. We believe ARE has a broader usage other than privacy-preserving collaborative filtering. For example, community detection [22], location-based services (checking if two users are nearby) and other applications which require a balance between efficiency and privacy in matching low-entropy (in a cryptographic sense) secrets.

Finally, we implement our scheme and evaluate its performance. We show that our solution is very efficient for practical use.

1.2 PAKE and Privacy-Preserving Matchmaking

After formulating the core functionality in this way, **password-based authenticated key exchange** (PAKE) [6] appears to be useful. It enables several parties holding a shared low-entropy password to securely establish a cryptographically strong session key. The major distinctive feature of PAKE is that, a secure PAKE protocol should withstand online dictionary attack, *i.e.*, one interaction of the protocol can only eliminate at most one possibility from the "passwords dictionary" (*i.e.*, one candidate in the password space). The security of a PAKE is usually established by upper bounding the probability of success by any adversary (under a certain formulation of security game between a challenger and an adversary) by something similar to $\frac{k_1}{2^\lambda} + \frac{k_2}{|\mathcal{M}|}$, where k_1 and k_2 are the number of attempts (modeled by "queries" either to the challenger or to the random oracle [7]) performed by the adversary, λ is a security parameter, and $\mathcal{M}$ is the password space. We will also formulate the security of our ARE in a similar vein.

Recently, Shin and Gligor [36] proposed a matchmaking protocol with enhanced privacy features. A matchmaking protocol enables two protocol participants holding the same "wish" (which may not have high entropy) to anonymously authenticate each other when their wishes match. Their protocol is based on PAKE [6,25,26]. To see how this protocol might be potentially useful in PPCF system, we start with the assumption that all users in the system are partitioned into well-defined "interest groups" according to their user-profiles. Given a particular user-profile, one could effectively determine which interest group it belongs to. In order to identify similar users, one could run the matchmaking protocol using the identifier of "interest group" as the "wish".

The major drawback of this approach is that the protocol is inherently interactive, as the underlying primitive PAKE is interactive. To the best of our

knowledge, there exists non-interactive AKE (which may be applicable on even weak mobile devices [44]), but not non-interactive PAKE. Even worse, this approach actually requires interaction between every pair of two parties, as that is the functionality supported by the underlying protocol of Shin and Gligor. Also, to maintain a certain level of authenticity (which is not a must in the PPCF setting), this matchmaking protocol requires the existence of a semi-trusted matchmaker, who is responsible for maintaining a list of pseudonyms of all valid users (and revoking misbehaved user's pseudonym if necessary). This semi-trusted matcher itself is a potential single-point of failure, which we tried to avoid. As a result, this approach is not that appropriate for our application.

2 Related work

2.1 Privacy-preserving Collaborative Filtering

Server based PPCF. To the best of our knowledge, privacy-preserving collaborative filtering was first formulated in the centralized server setting [32–34]. A typical scenario would be the following: privacy-concerned end-users want to obtain useful personalized recommendations from a untrusted service provider, but they are unwilling to compromise too much of their privacy. The service provider collects private data from different end-users and run a centralized CF calculator to generate user-specific recommendations.

In order to protect users' privacy, the general strategy adopted in the centralized setting [32–34] is to let users perturb their data before sending it to the service provider. Various perturbative techniques have been proposed to achieve privacy. For example, actual ratings could be randomized by noise addition [33], fake ratings could be inserted [34], and actual sensitive ratings could be suppressed (deleted) [32]. The high level idea underlying all these is that the untrusted server could only know a vague user-profile, and the noise level serves as a tunable parameter trading off recommendation accuracy for user privacy. Unfortunately, various studies [23, 24, 48] have shown that basic randomization techniques are not sufficient in many practical scenarios, where the adversary may possess some auxiliary information about target user.

To better quantify the level of privacy that could be obtained using randomized techniques, McSherry *et al.* [28] also considered the notion of differential privacy [15]. Yet the security model is slightly different since it only protect end-users' privacy against other curious users as well as outsiders, meaning that the centralized server is still trusted. It remains an interesting question on how to enable differential privacy against a untrusted centralized server.

Decentralized PPCF. Early work of P2P collaborative filtering (date back to 2002 [11]) relies on secure multi-party computation and homomorphic encryption. The most significant limitation of it (and follow-up work [1, 4]) is the high overhead due to the use of computationally expensive cryptographic tools.

Recently quite a few lightweight peer-to-peer protocols [8, 9, 29, 30, 37] have been proposed for identification of similar users with different level of privacy. Earlier approaches [8,9] require the end-users to broadcast their obfuscated profile with noise injected to find out similar users and then ask them for recommendations. The limitation of these systems is that individual profiles are essentially exposed in plaintext. Shokri *et al.* [37] addressed this problem by classifying users' profiles into two types, namely, offline and online. Online profile, being only a subset of offline one, is stored in an untrusted server to generate recommendations. Users keep their offline profile secretly but they will communicate with other users to aggregate offline profiles distributively. Online profiles are updated periodically by synchronizing with offline profiles. This can be seen as limiting the exposure by splitting the process into two stages. Nandi *et al.* [29,30] introduced the use of non-colluding decentralized middleware to enhance privacy. In a nutshell, these works either build on a weaker privacy model or rely on addition parties.

2.2 Cryptographic Approaches

The simplest approach to identify similar users is that, every participant broadcasts her profile in plaintext in a P2P network. Upon receiving other participant's profile, user privately decides whether this profile is similar to hers or not. Thus, users could generate recommendations using collected similar profiles. Each user might want to encrypt their profile for preserving their privacy. We have different options here. If symmetric key is used, the whole network needs to share the same key, which is clearly not a good solution. If the recipients' keys are used, a large amount of ciphertexts needs to be sent, or a large amount of computation (in the order of the size of the whole network) is needed to perform broadcast encryption (not to say most broadcast encryption schemes require a setup stage and is not possible in P2P setting.) The final option is to encrypt their profile using their own key. But it does not allow any comparison.

A few variants of public-key encryption may look potentially useful. However, they are not designed for our specialized purpose, so they are not efficient enough and may exhibit shortcomings in our application. More importantly, they cannot enforce *asymmetry* in computation times between an honest test and a malicious dictionary attack. We will elaborate one by one below. To this end, we believe new ideas are needed to develop the "right" cryptographic primitive for PPCF.

Probabilistic Public-Key Encryption with Equality Test (PKEET). PKEET [46] allows anyone to test whether two ciphertexts c_1, c_2 (possibly generated using different public keys) are encrypting the same message. Although it is primarily targeted for searchable encryption and encrypted data partition, PKEET appears to satisfy our functional requirement. Specifically, every user waits for the PKEET-encrypted profiles from others for comparison.

There are two major drawbacks. First, the test algorithms of all existing PKEET schemes [38, 39, 46] are implemented using bilinear map, which is not

that computationally efficient to process a large amount of ciphertexts. Second, PKEET allows *anyone* to test if two ciphertexts come from the same possibly *unknown* message. Public nature of the test means there is nothing differentiating an adversary from honest users. On the other hand, an adversary, without any knowledge of any profile, can just grab the ciphertexts from two different users and test if they correspond to the same profile.

Public-Key Encryption with Non-interactive Opening (PKENO). In PKENO [14, 16], opening refers to the decryption. PKENO allows one holding the key pair (pk, sk) to provide non-interactively to *any third party* a "proof" about a ciphertext c. Then, anyone with this proof can verify if c is indeed an encryption of a certain plaintext m under pk. For PPCF, every user can broadcast an encryption of her own profile together with a proof, then run a verification procedure locally to see if the received profiles will be opened to their own profile.

PKENO suffers from drawbacks similar to those of PKEET. The most efficient instantiation of PKENO [16] still requires eight modular exponentiations plus some other computations for verification. The non-interactive proof can be used by both an honest user and an adversary. Even worse, the proofs in many instantiations [16] can actually served as decryption keys, *i.e.*, attaching the proof simply reveals the message to everyone.

Plaintext-Checkable Encryption (PCE). PCE [10] is a randomized public-key encryption scheme that allows everyone to check its plaintext, *i.e.*, without the secret key, anyone can check if c is encrypting a plaintext m.

The original work of Canard *et al.* [10] showed how to transform any probabilistic public-key encryption scheme (and possibility its generalization like identity-based encryption) into a PCE, in the random oracle model. The basic idea is that the randomness ρ used in PKE to create a ciphertext c is derived from message m and random bit-string r ($\rho \leftarrow \mathcal{H}(m||r), r \leftarrow \{0,1\}^{\lambda}$), and r is also sent along with ciphertext c (for a certain security parameter λ). Given $c||r$, anyone holding the message m could then reproduce the randomness ρ and re-encrypt m to get c' herself. The remaining plaintext-check procedure is a simple equality check of $c' \stackrel{?}{=} c$.

Recall that our design goal is to enable an honest user with the knowledge of the hidden message (*i.e.*, the profile) to be able to perform the checking procedure faster than an adversary without a specific candidate m in mind. That appears to be not possible for their generic construction when it is instantiated by existing efficient probabilistic public-key encryption schemes such as ElGamal. In more details, the most expensive operation will be modular exponentiation, yet the exponent is unknown without the knowledge of the randomness ρ in the ciphertext, *i.e.*, the knowledge of the m does not play an important role here for possible acceleration of the checking procedure. Another important difference is that we do not require the decryption functionality supported by PCE. As a result, we could design simpler and hopefully more efficient schemes.

Deterministic Encryption (DE). Deterministic public key encryption (or DE for short), formalized by Bellare, Boldyreva, and O'Neill [5], has been a hot topic recently, as it provides an alternative when randomized encryption [20] has inherent drawbacks. DE finds its application in fast searching on encrypted data, or in scenarios where length-preserving ciphertexts are desirable.

DE can be used to realize PPCF, but apparently an offline dictionary attack of preparing DE's of all possible profiles can be launched. Standard salting helps. Yet that also hinders honest users as they need to use the different salt appended with the ciphertext for trail encryption and testing. Finally, as PCE, our PPCF application does not need decryption and a simpler scheme may suffice.

Fully Homomorphic Encryption (FHE). FHE is a powerful tool that allows secure evaluation over ciphertext. Secure instantiation of FHE is not known until 2009, when Gentry published his seminal work [17]. There are improvements in efficiency [18, 41], but it is still a bit far from practical for many applications.

Secure Multi-party Computation (SMC). SMC was first formalized by Yao [47] and Goldreich *et al.* [19], as a method for a group of mutually distrustful parties to jointly compute a function f on their private inputs. Some early work of PPCF [1, 11] used SMC as the underlying tool. They suffer from a high computation overhead, and they do not support dynamic user joining/leaving.

3 Preliminaries and Definitions

Here, we briefly review some basics about collaborative filtering and LSH, develop the notations for the rest of the paper, and lastly, define our new primitive.

3.1 Collaborative Filtering

In general, collaborative filtering (CF) algorithms can be broadly classified into two types: memory-based and model-based [42]. Our system only supports memory-based CF algorithms but we briefly mention model-based ones for completeness. We note that it is a challenging open question to support efficient privacy-preserving model-based CF, for the reason we will explain shortly.

Memory-based CF relies on pairwise statistical correlation. If two users have similar rating patterns according to existing records, they are likely to have similar opinion for some other items. We denote the rating from user u for an item i by r_{ui}, the set of all ratings of user u by a vector $\boldsymbol{r}_u$, the set of items rated by user u by S_u. There are many ways to define similarity, such as cosine similarity and Jaccard similarity. The cosine similarity of two vectors $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ is defined by $\mathsf{SIM}(r_1, r_2)_{\mathsf{cos}} = \frac{\boldsymbol{r}_1 \cdot \boldsymbol{r}_2}{\|\boldsymbol{r}_1\| \cdot \|\boldsymbol{r}_2\|}$. Similarly, the Jaccard similarity of two sets S_1 and S_2 is defined by $\mathsf{SIM}(S_1, S_2)_{\mathsf{Jac}} = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$.

Let set N be the top-k users most similar to user u and who also rated item i. Using ratings from these users, we could predict user u's rating for item i in many ways. We list two possible predictions as follows:

$$r_{ui} = \frac{1}{k} \sum_{u' \in N} r_{u'i} \text{ or } r_{ui} = \sum_{u' \in N} \frac{\mathsf{SIM}(u, u')}{\sum_{u' \in N} |\mathsf{SIM}(u, u')|} r_{u'i}$$

where $\mathsf{SIM}(u, u')$ denote some similarity metric like cosine similarity or Jaccard similarity defined above.

Model-based CF, as the name implies, performs filtering by modelling the global structures of users' ratings, instead of maintaining memory of users' rating. Important algorithms of this type include singular value decomposition (SVD), cluster analysis, Bayesian network, *etc.* Comparing with memory-based algorithms, model-based ones in general have better prediction performance but they are more computationally expensive. Yet, these algorithms require an overview of all users' ratings, which make it challenging to preserve privacy without using heavyweight cryptographic machineries, such as FHE and SMC.

3.2 Locality Sensitive Hashing

Our system relies on Locality Sensitive Hashing (LSH) [2] to allow individual user to identify similar users locally, which we briefly review below.

A family $\mathcal{F}$ of LSH functions operates on a collection of objects. The most interesting and important property of an LSH function is that, similar objects are more likely to be hashed to the same bucket. Formally, let $\mathsf{SIM}(x, y)$ denote some similarity metric defined on the collection of objects, an LSH family satisfies:

$$\Pr_{h \in \mathcal{F}}[h(x) = h(y)] = \mathsf{SIM}(x, y).$$

Similar to the recent PPCF systems [13, 30], we consider the following LSH of Charikar [12] that is defined over cosine similarity. First, pick k random vectors, with components drawn independently from a Gaussian distribution ($\mu = 0, \sigma = 1$). To calculate the LSH digest of a user-profile $\boldsymbol{r}$, we need to calculate the dot product of $\boldsymbol{r}$ with each random vector $\boldsymbol{v_i}$, namely $\boldsymbol{v}_i \cdot \boldsymbol{r}$. The i^{th} bit of the LSH digest $L(\boldsymbol{r})[i]$ is set to 1 if $\boldsymbol{v}_i \cdot \boldsymbol{r} > 0$, 0 otherwise.

3.3 Cryptographic Notations

A binary string is represented using lower case letters like x and $|x|$ denote its length. The i^{th} bit of x is $x[i]$ and $x[i, j]$ denotes $x[i] \dots x[j]$ for $1 \leq i \leq j \leq |x|$. If S is a finite set then $|S|$ denotes its size and $s \xleftarrow{\$} S$ denotes picking an element uniformly random from the set S. For $i \in \mathbb{N}$, we let $[i] = \{1, \dots, i\}$. We denote the security parameter by $\lambda \in \mathbb{N}$ and its unary representation by 1^λ.

Algorithms are polynomial time (PT) and randomized unless otherwise indicated. By $y \xleftarrow{\$} A(x_1, \dots; R)$ we denote running algorithm A on input $x_1, \dots$ using randomness R, and assigning the output to y. We may omit R for brevity.

Let $\mathbb{G}$ denote a group of order p, where p is a λ bit prime number, and g is a generator of $\mathbb{G}$. If m is a binary string of length less than or equal to λ, then we use *capital letter* M to denote some *efficient mapping* of m as a group element in $\mathbb{G}$. We do not expect any special property from this map. In particular, it does not need to be a cryptographic hash. We use $\mathcal{M}$ to denote the message space.

A family of hash functions $\mathsf{H} = (\mathcal{HK}, \mathcal{H})$ is a pair of PT algorithms, the second is deterministic. The key generation algorithm $\mathcal{HK}$ takes input 1^λ and returns a hashing key K_h. The hashing algorithm $\mathcal{H}$ takes K_h and a message m and returns its hash $H \leftarrow \mathcal{H}(K_h, m)$. In the security proof of our scheme, the hash function will be modelled as a random oracle [7].

3.4 Asymmetric Randomized Encoding

An Asymmetric Randomized Encoding scheme $\mathsf{ARE} = (\mathcal{P}, \mathcal{E}, \mathcal{T})$ is a tuple of three PT algorithms. The parameter generation algorithm $\mathcal{P}(\cdot)$ takes a security parameter 1^λ as input, and returns public parameter P. The encoding algorithm $\mathcal{E}(\cdot,\cdot)$ takes the public parameter P and a binary string m as input, returns $\mathcal{E}(P, m)$ as an encoding of m. $\mathcal{T}(\cdot,\cdot,\cdot)$ is the test algorithm that takes $P, m, \mathcal{E}(P, m')$ as input, outputs a boolean variable T depending on the relation of m and m'. Recall that the message uncertainty is the only thing differentiates an attacker from an honest user. We formalize test correctness and two security requirements for an ARE scheme as follows. For brevity, the probabilistic generation of the public parameter $P \xleftarrow{\$} \mathcal{P}(1^\lambda)$ is omitted from the description of the probabilities below.

Test correctness requires that it is universally possible to check whether the preimage of $\mathcal{E}(P, m')$ equals to m with overwhelming probability. Formally,

$$\Pr[\mathcal{T}(P, \mathcal{E}(P, m'), m) = \textit{'True'} \,|m = m'] > 1 - \delta,$$
$$\Pr[\mathcal{T}(P, \mathcal{E}(P, m'), m) = \textit{'False'} \,|m \neq m'] > 1 - \delta,$$

where δ is negligible in λ.

Privacy requires that it is difficult to recover m only given $\mathcal{E}(P, m)$. Formally, we say that ARE satisfies *privacy* if the advantage of an adversary A against privacy satisfies

$$\mathbf{Adv}^{\mathsf{Privacy}}_{\mathsf{ARE},A} = \Pr[m' = m | m' \xleftarrow{\$} A(P, \mathcal{E}(P, m))] \leq \frac{k_1}{2^\lambda} + \frac{k_2}{|\mathcal{M}|}$$

where k_1 and k_2 are polynomial in λ.

Unlinkability requires that it is difficult to guess whether two encodings come from the same message or not, when the messages are unknown. Formally, unlinkability is defined via the following game. The challenger chooses a pair of distinct messages m_0 and m_1, and a bit b, uniformly at random. The adversary A has polynomially-many access to encoding oracles $\mathcal{O}_{\mathcal{E}_0}(\cdot)$ and $\mathcal{O}_{\mathcal{E}_1}(\cdot)$, which returns encodings of m_0 and m_1 respectively. A is also given an encoding of

m_b, *i.e.*, $\mathcal{E}(m_b)$. Finally, A outputs a bit b'. We say that ARE is *unlinkable* if the advantage of an adversary breaking the Unlink game, denoted by $\mathbf{Adv}^{\mathsf{Unlink}}_{\mathsf{ARE},A}$, satisfies

$$\begin{aligned}\mathbf{Adv}^{\mathsf{Unlink}}_{\mathsf{ARE},A} =& |\Pr[b' = b | b \xleftarrow{\$} \{0,1\}, m_0 \xleftarrow{\$} \mathcal{M}, m_1 \xleftarrow{\$} \mathcal{M} \setminus \{m_0\},\\ & b' \xleftarrow{\$} A^{\mathcal{O}_{\varepsilon_0}, \mathcal{O}_{\varepsilon_1}}(P, \mathcal{E}(P, m_b))] - \frac{1}{2}|\\ \leq & \frac{k_1}{2^\lambda} + \frac{k_2}{|\mathcal{M}|}\end{aligned}$$

where k_1 and k_2 are polynomial in λ.

Note that encodings by PKEET, or any deterministic scheme like DE, would be insecure under this definition, due to the efficient algorithm for deciding if two ciphertexts are encrypting the same plaintext.

4 System Model

We follow the system model of Berkovsky *et al.* [8]. We assume that users are organized in a purely P2P manner [11]. Within this P2P network, users could freely contact any other users who also joined the system. Such a system could be built using existing technologies (*e.g.*, [3]).

4.1 Profile Representation and Basic System Setup

Every end-user in the system is the holder of their own private user-profiles. Without loss of generality, we follow the existing representation [30] of user-profiles in the form of a list of ⟨key, value⟩ pairs, where keys could represent any commodity like books, movies, or other categories of goods, and values represent the interest level to the commodity corresponding to the key. Note that it is easy to transform this ⟨key, value⟩ pair representation into simple vector representation, as long as the size of possible key set is fixed and the positions of different keys are determined.

We assume that all participants have previously agreed on a consistent encoding of user-profile (*e.g.*, the range of interest level). Also, they have agreed to use a selected LSH function, and a specific ARE scheme. In other words, the algorithms to use and their public system parameters are fixed for all users.

We emphasize that we do not assume any trusted or semi-trusted third party to support our system (although they could be easily added to our system to support more features). In particular, all the system parameters mentioned above can be generated without using any secret keys or trapdoor. Using this setting, we eliminate any trust issue of a single point in terms of privacy and availability.

4.2 Entities and Threat Model

End-users in the system want to obtain useful information from other participants. All users broadcast a short (comparing with the potentially long user-profile) randomized "identifier", in the form of encoded LSH digest, to their neighbourhood. Looking ahead, this "identifier" only leaks just enough information for other users to check whether the underlying LSH digest is similar to theirs or not, and no more information is leaked. We will also show how to identify similar users non-interactively in the following Section 6.

End-users in our system do not trust each other. In particular, they are only willing to expose their profile to those who are holding similar user-profile. That is to say, from an arbitrary user u's perspective, all other users are divided into two groups: similar users and dissimilar users, and these two groups are treated differently. Every dissimilar users and their coalition is treated as an adversary in our threat model, which is interested to gather global statistics of all users.

We further assume that users are honest-but-curious. That is to say, they will not deviate from the protocol specification but they want to gain more information by analyzing protocol transcript offline. Naturally, an adversary can always prepare a "fake" profile or even inject many such profiles to the system. Authenticity of profile and sybil-resistance are out of scope of this paper and can be dealt with additional measures (*e.g.*, [40]). Our goal is to protect against unnecessary information leakage of users. In particular, the best an adversary can do is to launch a dictionary attack per each encoded profile obtained.

5 Instantiation of ARE

5.1 Proposed Construction

The **parameter generation algorithm** $\mathcal{P}(1^\lambda)$ randomly selects a λ-bit prime p, a prime order group $\mathbb{G}$ of order p, and two independent hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$ and $H_2 : \{0,1\}^\lambda \to \{0,1\}^\lambda$. Parameter $P = (p, \mathbb{G}, H_1, H_2)$ is generated as output. Here, $\mathbb{Z}_p^*$ is just $[p]$. All users joining our system are assumed to have agreed on a set of public parameters, thus we omit P below as input for brevity.

The **encoding algorithm** $\mathcal{E}$ takes input a λ-bit message m, selects a λ-bit random string r uniformly, returns $\mathcal{E}(m) = (c_1, c_2) = (M^{H_1(r)}, H_2(m) \oplus r)$ as output, where M is some efficient mapping of m to a group element in $\mathbb{G}$.

The **test algorithm** $\mathcal{T}$ takes input m and $\mathcal{E}(m')$. It first parses $\mathcal{E}(m')$ as (c_1, c_2), then compute $r = c_2 \oplus H_2(m)$, and finally return $M^{H_1(r)} \stackrel{?}{=} c_1$ as output.

5.2 Pre-computation of Honest Users

It is trivial to see that our construction satisfies test correctness. Now we show the "asymmetry" of our construction. For an honest user holding M, she could perform some precomputation [27] by preparing $\hat{M}_i = M^{2^i}$ for $i = 0, 1, \ldots, |p|-1$. Upon receiving some encoding $\mathcal{E}(m') = (c_1, c_2)$, she first recovers r by $H_2(m) \oplus c_2$.

Parameter Generation $\mathcal{P}(1^\lambda)$
Randomly select p, group $\mathbb{G}$ of order p, and hash functions $H_1 : \{0,1\}^* \to \mathbb{Z}_p^*$,$H_2 : \{0,1\}^\lambda \to \{0,1\}^\lambda$. Return $P = (p, \mathbb{G}, H_1, H_2)$.

Encoding $\mathcal{E}(m)$
Randomly select $r \xleftarrow{\$} \{0,1\}^\lambda$. Map m to $M \in \mathbb{G}$. Return $(c_1, c_2) = (M^{H_1(r)}, H_2(m) \oplus r)$.

Testing $\mathcal{T}(m, \mathcal{E}(m'))$
Parse $\mathcal{E}(m')$ as (c_1, c_2). Map m to $M \in \mathbb{G}$. Compute $r = c_2 \oplus H_2(m)$. Return $M^{H_1(r)} \stackrel{?}{=} c_1$.

Fig. 1. Our Proposed Construction

Let $R \subset [|p|]$ be the set of indices such that $r[i] = 1$. Instead of computing exponentiation $M^{H_2(r)}$ directly, she only needs to compute a few multiplications, namely $\prod_{i \in R} \hat{M}_i$, which are quite minimal.

We would like to point out that this pre-computation step does not reduce the asymptotic complexity of modulo exponentiation. Suppose the base is n-bit and the exponent is λ-bit, using this pre-computation trick the overall complexity of modulo exponentiation remains $O(\lambda c(n))$, where $c(n)$ is the complexity of multiplication. But with proper implementation, this trick could still improve upon the standard repeated squaring algorithm by some constant factor, which could make a big difference when we are dealing with big data (of many candidate profiles). To see this, notice that an honest user is only interested in filtering out similar messages. Most of the computation are raising her M up to a certain power. With the above trick, each exponentiation requires only $\lambda/2$ multiplications on average instead of λ multiplications using the repeated squaring algorithm.

On the other hand, when the adversary's goal is recovering the hidden message from a *specific* encoding or see if any two encodings actually correspond to the same message, offline dictionary attack is needed to exhaust all possibilities in the message space $\mathcal{M}$, which we will show shortly afterwards. In other words, those without a specific m in mind cannot enjoy pre-computation.

5.3 Security Proofs

The following theorem asserts that our scheme satisfies our definition of privacy.

Theorem 1. *Our scheme in Figure 1 satisfies privacy and unlinkability (defined in Section 3) in the random oracle model with H_1 and H_2 being random oracles.*

Proof. The adversary A is given $\mathcal{E}(m) = (c_1, c_2) = (M^{H_1(r)}, H_2(m) \oplus r)$ where m and r are both chosen uniformly at random. Since H_1 and H_2 are both random

oracles, both $M^{H_1(r)}$ and $H_2(m) \oplus r$ should be totally random from adversary's point of view, leaking no information about r and M, unless either $H_1(r)$ or $H_2(m)$ has been queried by the adversary.

As $\mathcal{E}(m) = (c_1, c_2)$ leaks no information about r and M, an adversary could only make random queries to the two random oracles $H_1(\cdot)$ and $H_2(\cdot)$. Suppose the adversary is only given k_1 and k_2 accesses to the two random oracles respectively, where k_1 and k_2 are positive integers. Then the probability that such queries collide with r and m is $P_1 = \frac{k_1}{2^\lambda}$ and $P_2 = \frac{k_2}{|\mathcal{M}|}$ respectively. By union bound, the probability that collision happens on either random oracle (which equals $\mathbf{Adv}^{\mathsf{Privacy}}_{\mathsf{ARE},A}$) is

$$\mathbf{Adv}^{\mathsf{Privacy}}_{\mathsf{ARE},A} = P \leq P_1 + P_2 = \frac{k_1}{2^\lambda} + \frac{k_2}{|\mathcal{M}|} \tag{1}$$

which concludes our proof for *privacy*.

The proof for *unlinkability* is in spirit very similar to the above proof. At the very beginning of the game, the challenger secretly picks $h_0, h_1 \xleftarrow{\$} \{0,1\}^\lambda$ and sets $H_2(m_0) = h_0, H_2(m_1) = h_1$ in its internal table. For every query to either encoding oracles, or for supplying $\mathcal{E}(m_b)$ to the adversary, the challenger always responds by returning a pair of strings randomly chosen from the appropriate domain. All these are valid responses unless collision occurs. With no collision, the adversary can only have $\frac{1}{2}$ chance in guessing the bit b correctly.

Notice that from adversary's point of view, for either case of m_0 or m_1, all received encodings can be interpreted in the correct encoding format as $(M^{r_1^*}, c_2) = (M^{H_1(c_2 \oplus H_2(m))}, c_2 \oplus H_2(m) \oplus H_2(m)) = (M^{H_1(r_2^*)}, r_2^* \oplus H_2(m))$, where $r_2^* = c_2 \oplus H_2(m)$, unless the adversary has queried $H_1(c_2 \oplus H_2(m))$ before. However, $c_2 \oplus H_2(m)$ is totally random because c_2 is chosen uniformly random and the adversary do not know $H_2(m)$. As a result, the adversary could only make random queries to $H_1(\cdot)$ and $H_2(\cdot)$. If it so happened that the adversary has queried $H_2(m_0)$ or $H_2(m_1)$, she could easily distinguish $\mathcal{E}(m_0)$ from $\mathcal{E}(m_1)$ by recovering r_2^* and further querying $H_1(r_2^*)$. This happens with probability at most $\frac{2k}{|\mathcal{M}|}$. On the other hand, if the adversary has queried $H_1(c_2 \oplus h_0)$ or $H_1(c_2 \oplus h_1)$ before, then the challenger has no freedom to set r_1^* to be the hash value. This happens with probability $\frac{2k}{2^\lambda}$, where k is the number of queries made by the adversary. To conclude, such unlikely event happens with probability at most $\frac{2k}{2^\lambda} + \frac{2k}{|\mathcal{M}|}$ by union bound. Thus we have $\mathbf{Adv}^{\mathsf{Unlink}}_{\mathsf{ARE},A} \leq \frac{2k}{2^\lambda} + \frac{2k}{|\mathcal{M}|}$, which concludes our unlinkability proof. □

5.4 Discussion

One may ask why we chose to model security using one-wayness. We note that it is impossible to achieve any security against chosen-plaintext attack (CPA) or alike formulation because we mandate the "test correctness" requirement. If an adversary is given the ability to test whether a certain encoding corresponds to a certain message, he could trivially win any form of CPA game. Similar

issues have been discussed in the literature of related cryptographic primitives like PKEET and PKENO [10,46]. Also, there is no guarantee that m would have high entropy. Thus an adversary could always launch offline-dictionary attack. However, we insist that offline-dictionary attack should be the "best-possible" attack, namely there would not be any "smarter" algorithm. This fact is captured by the right hand side of our probability guarantee in Equation (1). We believe that we have targeted for "best possible" security in our application.

6 Privacy-Preserving Filtering

We describe our system from the perspective of a user u, from joining the system to actually obtaining useful recommendations, via the following four stages.

Preparation of User's Own Profile. User u prepares her own profile, and applies the LSH function described in Section 3 to obtain a k-bit digest, where k is some multiple of λ. She then chops this digest into blocks of λ-bit long strings $m_1, m_2, \ldots$ and encodes these strings to $\mathcal{E}(m_1), \mathcal{E}(m_2), \ldots$ using our ARE scheme.

Broadcasting of Encoded Profile. Every user broadcasts the encodings to their neighbourhood. This procedure is repeated periodically for informing others about one's existence. It is easy to see that a new user can join the system freely. On the other hand, a time-to-live value can be attached to expire inactive users.

Identification of Matching Profiles. Other users in the system are also sending their profiles in the network periodically, so user u may receive such packets. Upon receiving encodings from others, user u decides locally which users are similar to her. This is done by running algorithm $\mathcal{T}$ of our ARE on each block of λ-bit string. Specifically, if there are at least t such blocks are equal to her own digest, then she considers this user to be similar to her. We call the parameter t as similarity threshold, and it is completely decided by user u herself. The more identical blocks, the more similar they are.

Actual Recommendation Stage. After some time, user u should be able to compile a list of similar users with their LSH-digest, and their similarity degree. User u contacts similar users in order to exchange user profile secretly. The policy of choosing who to contact is again totally up to user u herself. For example, she might decide to choose the top 10 similar users, or she might decide to choose 20 random similar users in order to have better diversity. After collecting enough responses from other similar users, user u runs a memory based collaborative filtering algorithm (see Section 3) to generate recommendations herself.

How to Contact Users Securely. There are various ways to implement the required secure channel. They might just run a Diffie-Hellman key exchange protocol; or they might decide to run a PAKE protocol with the "password" being the common bits of their LSH digest which represents their wishes.

The exact information being exchanged between similar users are also up to their own choice. They can send an obfuscated version of their profile to others.

7 Evaluation

We implement ARE using Crypto++ v5.63. The system we use for performing our time analysis is Acer Aspire V5-473G, with 8GB memory, and Intel Core i7-4500U 1.8GHz with Turbo Boost up to 3.0GHz, We use the standard SHA3 hash algorithm to instantiate hash functions H_1 and H_2[1].

We measure the performance in identifying matching profiles using various parameter settings listed below, where λ refers to the security parameter and k denotes the length of LSH digest. Thus $\frac{k}{\lambda}$ is the number of blocks. For each k, λ combination, a fixed m is randomly chosen, and $10,000$ encoded random profiles $\mathcal{E}(m_r)$ are generated to execute $\mathcal{T}(m, \mathcal{E}(m_r))$. All the obtained figures are the averaged result of $10,000$ such executions. The times are measured in milliseconds. Note that we run the experiment using a commodity laptop with not-yet optimized code. From these figures, it is clear that our scheme is very efficient and practical.

For the choice of parameters k, λ and t, intuitively, a larger λ enlarges the size of $\mathcal{M}$, which in turn means better privacy. On the other hand, according to the property of LSH, it becomes less likely to find an exact match. To overcome this, we could choose a bigger k to reduce the probability that none of the blocks matches, at the cost of computation cost and communication overhead. Thus, we can tune our parameters for different levels of privacy, efficiency, and usability.

Setting	Time (ms)	Setting	Time (ms)
$\lambda = 64, k = 256$	0.2293	$\lambda = 128, k = 256$	0.2254
$\lambda = 64, k = 512$	0.4603	$\lambda = 128, k = 512$	0.4566
$\lambda = 64, k = 1024$	0.9093	$\lambda = 128, k = 1024$	0.8633
$\lambda = 64, k = 2048$	1.8152	$\lambda = 128, k = 2048$	1.6962

Fig. 2. Performance Evaluation for $10,000$ Profiles

Our system computes a long LSH digest for each profile, chops the LSH digest into blocks, and uses the number of identical blocks to detect similar profiles.

[1] We prepend dummy strings S_1 and S_2 to the input to instantiate two hash functions. The first λ-bit output of SHA3 is picked as output, *i.e.*, $H_i(m) = \text{SHA3}(S_i||m)[0, \ldots, \lambda - 1]$ for $i \in [1, 2]$. For $H_1(m)$, there exists a small probability that the output is larger than $p - 1$. If that occurs, we re-hash the result until it fits.

This approach is somewhat different from previous schemes [12, 13, 35] where similarity is measured by the number of identical bits directly. To demonstrate that our modification works reasonably well, we conducted the following additional experiment. We constructed a random Netflix user profile p consisting of 200 ratings. Each rating is chosen from a Gaussian distribution with $\mu = 3.8$ and $\sigma = 1$ (according to the statistics of Netflix dataset [21]). Ratings are rounded to the nearest integers (and confined within $\{1, \cdots, 5\}$) to fit with the Netflix rating format. Then three other Netflix profiles r_1, r_2, r_3 are randomly constructed to represent "dissimilar profile", "similar profile" and "very similar profile" compared with p. These profiles are created by adding zero-mean Gaussian noise to p with different variances ($\sigma = 0.5, 0.4, 0.3$ respectively). An 1024-bit digest for each of these four profiles are computed using the LSH algorithm described in Section 3.2. We calculated the number of different bits and also the number of identical blocks compared with the digest of p (block length is 64-bit). The above experiment was repeated 500 times by choosing different LSH functions to obtain the following averaged numbers. The result is summarized in Fig 3, from which we can conclude that, while the LSH digests of r_i's are all similar to those of p, there exists a clear distinction among "dissimilar profile", "similar profile" and "very similar profile" via our block-wise comparison (see the last row).

	Dissimilar Profile r_1	Similar Profile r_2	Very Similar Profile r_3
Noise Variance	0.5	0.4	0.3
# of Different Bits	44.672	32.042	18.592
# of Identical Blocks	1.08	2.13	5.04

Fig. 3. Performance Evaluation of LSH Algorithm based on ARE ($k = 1024, \lambda = 64$)

8 Conclusion

In this paper, we present *asymmetric randomized encoding*, a simple yet novel cryptographic primitive that could be used for efficient privacy preserving filtering. We define appropriate security notion for this primitive, and provide a simple and efficient construction. We prove the security of this construction in the random oracle model and evaluate its performance. We also describe how to use this primitive to build a practical peer-to-peer privacy-preserving collaborative filtering system.

References

1. W. Ahmad and A. A. Khokhar. An Architecture for Privacy Preserving Collaborative Filtering on Web Portals. In *IAS*, pages 273–278. IEEE Computer Society,

2007.
2. A. Andoni and P. Indyk. Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Commun. ACM*, 51(1):117–122, 2008.
3. S. Androutsellis-Theotokis and D. Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.
4. A. Basu, J. Vaidya, H. Kikuchi, and T. Dimitrakos. Privacy-preserving Collaborative Filtering for the Cloud. In C. Lambrinoudakis, P. Rizomiliotis, and T. W. Wlodarczyk, editors, *CloudCom*, pages 223–230. IEEE, 2011.
5. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and Efficiently Searchable Encryption. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 535–552. Springer, 2007.
6. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In B. Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer, 2000.
7. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.
8. S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci. Enhancing Privacy and Preserving Accuracy of a Distributed Collaborative Filtering. In J. A. Konstan, J. Riedl, and B. Smyth, editors, *RecSys*, pages 9–16. ACM, 2007.
9. M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. The Gossple Anonymous Social Network. In I. Gupta and C. Mascolo, editors, *Middleware*, volume 6452 of *Lecture Notes in Computer Science*, pages 191–211. Springer, 2010.
10. S. Canard, G. Fuchsbauer, A. Gouget, and F. Laguillaumie. Plaintext-Checkable Encryption. In O. Dunkelman, editor, *CT-RSA*, volume 7178 of *Lecture Notes in Computer Science*, pages 332–348. Springer, 2012.
11. J. F. Canny. Collaborative Filtering with Privacy. In *IEEE Symposium on Security and Privacy*, pages 45–57. IEEE Computer Society, 2002.
12. M. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In J. H. Reif, editor, *STOC*, pages 380–388. ACM, 2002.
13. R. Chow, M. A. Pathak, and C. Wang. A Practical System for Privacy-Preserving Collaborative Filtering. In J. Vreeken, C. Ling, M. J. Zaki, A. Siebes, J. X. Yu, B. Goethals, G. I. Webb, and X. Wu, editors, *ICDM Workshops*, pages 547–554. IEEE Computer Society, 2012.
14. I. Damgård, D. Hofheinz, E. Kiltz, and R. Thorbek. Public-Key Encryption with Non-interactive Opening. In T. Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 2008.
15. C. Dwork. Differential Privacy: A Survey of Results. In M. Agrawal, D. Du, Z. Duan, and A. Li, editors, *TAMC*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2008.
16. D. Galindo, B. Libert, M. Fischlin, G. Fuchsbauer, A. Lehmann, M. Manulis, and D. Schröder. Public-Key Encryption with Non-Interactive Opening: New Constructions and Stronger Definitions. In D. J. Bernstein and T. Lange, editors, *AFRICACRYPT*, volume 6055 of *Lecture Notes in Computer Science*, pages 333–350. Springer, 2010.
17. C. Gentry. Fully Homomorphic Encryption using Ideal Lattices. In M. Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
18. C. Gentry, S. Halevi, and N. P. Smart. Fully Homomorphic Encryption with Polylog Overhead. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.

19. O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In A. V. Aho, editor, *STOC*, pages 218–229. ACM, 1987.
20. S. Goldwasser and S. Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
21. I. Grigorik. Dissecting the Netflix Dataset - igvita.com. Last accessed on 2014-09-12.
22. P. Hu, S. S. M. Chow, and W. C. Lau. Secure Friend Discovery via Privacy-Preserving and Decentralized Community Detection. In *ICML 2014 Workshop on Learning, Security and Privacy*, 2014. Full version appear at http://arxiv.org/abs/1405.4951.
23. Z. Huang, W. Du, and B. Chen. Deriving Private Information from Randomized Data. In F. Özcan, editor, *SIGMOD Conference*, pages 37–48. ACM, 2005.
24. H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the Privacy Preserving Properties of Random Data Perturbation Techniques. In Wu et al. [45], pages 99–106.
25. J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. In B. Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 475–494. Springer, 2001.
26. J. Katz, R. Ostrovsky, and M. Yung. Efficient and Secure Authenticated Key Exchange using Weak Passwords. *J. ACM*, 57(1), 2009.
27. J. K. Liu, J. Baek, J. Zhou, Y. Yang, and J. W. Wong. Efficient Online/offline Identity-based Signature for Wireless Sensor Network. *Int. J. Inf. Sec.*, 9(4):287–296, 2010.
28. F. McSherry and I. Mironov. Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders. In J. F. E. IV, F. Fogelman-Soulié, P. A. Flach, and M. J. Zaki, editors, *KDD*, pages 627–636. ACM, 2009.
29. A. Nandi, A. Aghasaryan, and M. Bouzid. P3: A Privacy Preserving Personalization Middleware for Recommendation-based Services. In *Hot Topics in Privacy Enhancing Technologies Symposium*, 2011.
30. A. Nandi, A. Aghasaryan, and I. Chhabra. On the Use of Decentralization to Enable Privacy in Web-scale Recommendation Services. In Sadeghi and Foresti [35], pages 25–36.
31. A. Narayanan and V. Shmatikov. Robust De-anonymization of Large Sparse Datasets. In *IEEE Symposium on Security and Privacy*, pages 111–125. IEEE Computer Society, 2008.
32. J. Parra-Arnau, D. Rebollo-Monedero, and J. Forné. A Privacy-Protecting Architecture for Collaborative Filtering via Forgery and Suppression of Ratings. In J. García-Alfaro, G. Navarro-Arribas, N. Cuppens-Boulahia, and S. D. C. di Vimercati, editors, *DPM/SETOP*, volume 7122 of *Lecture Notes in Computer Science*, pages 42–57. Springer, 2011.
33. H. Polat and W. Du. Privacy-Preserving Collaborative Filtering Using Randomized Perturbation Techniques. In Wu et al. [45], pages 625–628.
34. H. Polat and W. Du. Achieving Private Recommendations Using Randomized Response Techniques. In W. K. Ng, M. Kitsuregawa, J. Li, and K. Chang, editors, *PAKDD*, volume 3918 of *Lecture Notes in Computer Science*, pages 637–646. Springer, 2006.
35. A. Sadeghi and S. Foresti, editors. *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*. ACM, 2013.

36. J. S. Shin and V. D. Gligor. A New Privacy-Enhanced Matchmaking Protocol. *IEICE Transactions*, 96-B(8):2049–2059, 2013. Preliminary version appeared at NDSS 2008.
37. R. Shokri, P. Pedarsani, G. Theodorakopoulos, and J.-P. Hubaux. Preserving Privacy in Collaborative Filtering through Distributed Aggregation of Offline Profiles. In L. D. Bergman, A. Tuzhilin, R. D. Burke, A. Felfernig, and L. Schmidt-Thieme, editors, *RecSys*, pages 157–164. ACM, 2009.
38. Q. Tang. Public Key Encryption Schemes supporting Equality Test with Authorisation of Different Granularity. *IJACT*, 2(4):304–321, 2012.
39. Q. Tang. Public Key Encryption supporting Plaintext Equality Test and User-specified Authorization. *Security and Communication Networks*, 5(12):1351–1362, 2012.
40. D. N. Tran, J. Li, L. Subramanian, and S. S. M. Chow. Optimal Sybil-resilient Node Admission Control. In *INFOCOM*, pages 3218–3226. IEEE, 2011.
41. M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully Homomorphic Encryption over the Integers. In H. Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.
42. Wikipedia. Collaborative Filtering. `http://en.wikipedia.org/wiki/Collaborative_filtering`, 2014. Last accessed on 2014-09-12.
43. Wikipedia. Netflix Prize. `http://en.wikipedia.org/wiki/Netflix_Prize`, 2014. Last accessed on 2014-09-12.
44. T.-S. Wu and H.-Y. Lin. Non-Interactive Authenticated Key Agreement over the Mobile Communication Network. *MONET*, 18(5):594–599, 2013.
45. X. Wu, A. Tuzhilin, and J. Shavlik, editors. *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*. IEEE Computer Society, 2003.
46. G. Yang, C. H. Tan, Q. Huang, and D. S. Wong. Probabilistic Public Key Encryption with Equality Test. In J. Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 119–131. Springer, 2010.
47. A. C.-C. Yao. How to Generate and Exchange Secrets (Extended Abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.
48. S. Zhang, J. Ford, and F. Makedon. Deriving Private Information from Randomly Perturbed Ratings. In J. Ghosh, D. Lambert, D. B. Skillicorn, and J. Srivastava, editors, *SDM*, pages 59–69. SIAM, 2006.