A Private and Reliable Recommendation System for Social Networks

T. Ryan Hoens and Marina Blanton
Department of Computer Science and Engineering
University of Notre Dame
{thoens,mblanton}@cse.nd.edu

Nitesh V. Chawla
Interdisciplinary Center for Network Science and Applications
University of Notre Dame
nchawla@cse.nd.edu

***Abstract*—With the proliferation of internet-based social networks into our lives, new mechanisms to control the release and use of personal data are required. As a step toward this goal, we develop a recommendation system which protects the privacy of user answers while allowing them to learn an aggregate weighted average of ratings. Due to the use of social network connections, the querier obtains a more relevant and trustworthy result than what generic anonymous recommendation systems can provide, while at the same time preserving user privacy. We also give experimental performance results for our solution and several recently developed secure computation techniques, which is of independent interest.**

I. Introduction

Over the last decade social networking sites have increased in popularity. This popularity has brought many people together, increasing their ability to share information. While in general information sharing is desirable, some users might be concerned with the privacy implications of disclosing so much personal information online. Currently users have two main options: They can either refuse to enter the information they are uncomfortable disclosing, or they may limit access to the information via privacy controls provided by the social networking site. It is important to note, however, that in the second case the information is stored remotely, and thus control of the data is lost to the user.

While the current system is functional, a more desirable alternative would be to enable users to retain control over their data, while retaining utility to friends in the social network. Product reviews, for example, can benefit from a system that facilitates information sharing while maintaining participants' privacy. While some users might not be willing to disclose their (numerical) ratings of certain products (or the existence of a rating), they would still be able to give their opinion. Without input from a social network, users who had not tried a certain product would instead be forced to sample the product themselves, or read reviews from anonymous users to be able to make an intelligent decision about the product. Both of these options are suboptimal as they disregard the benefits of friendship relationships available. In particular, recommendations gathered through a social network are likely to be more useful than general systems due to similarities between the tastes of the querier and its network. Information gathered through such means is also likely to be more trustworthy as it relies on connections between individuals, and thus enabling users to query friends for their opinions would be beneficial.

In order to facilitate this, we propose a protocol whereby users are able to query a neighborhood of their social network in order to learn a recommendation. The rating is computed as a weighted average of individual ratings, with users closer to the querier (optionally) having more weight. During no point of the computation should any user (nor the underlying system) be able to learn any additional information. That is, only the querier learns the weighted average of ratings from its social network, while everyone else learns nothing. This should alleviate any privacy concerns users may have about disclosing their opinions. Furthermore, we provide a mechanism for claiming that no rating exists.

An additional substantial contribution of this work is the extensive implementation of our recommendation system integrated with the Facebook social networking site. To the best of our knowledge, this is the first implementation illustrating the performance of several highly non-trivial techniques for secure multi-party computation from prior literature. Such results are of independent interest and have high practical relevance.

II. Overview of Scheme

As previously stated, our goal is to compute a weighted average of ratings among a group whose individual values are considered private. In this section we provide a description of the computation performed during an invocation our scheme, a description of the security model, and an overview of the techniques. Building blocks are presented in Section III, the full scheme is given in Section IV, and Section V describes our implementation and the performance results.

A. Overview of Computation

We view the network as a hierarchical structure, where the user submitting the query acts as a "root" and the immediate friends as "children." The root can specify the distance to which the query should propagate. While in general any depth can be specified, for convenience we describe the computation using depth 2 (as we believe the results will be most useful to the root for low values of the depth). In our setup, we assume neither the object being rated nor the querier are sensitive, but all information regarding user ratings must remain secure.

978-0-7695-4211-9/10 $26.00 © 2010 IEEE
DOI 10.1109/SocialCom.2010.124

Initially, the root, R, wishes to learn the rating of an item, I. To do so, R first chooses a set of friends, F, who are trusted to determine an average rating. R then requests that each $f \in F$ computes a rating of I, where the range of possible ratings is specified as a subset of $\mathbb{N}^+$. Upon being queried for a rating of I, each $f \in F$ then queries its friends for their rating of I. Each rating is represented as the pair (s_i, w_i), where s_i is the rating and w_i is the weight. When a user i does not need to query her children (e.g., is at the bottom of the hierarchy), she merely returns her rating r_i as the value for s_i and sets the weight w_i to 1, or, alternatively, if user i does not have (or wish to provide) a rating for the item, the pair $(0, 0)$ is returned. Other users in the hierarchy compute their rating as a function of their own rating and ratings of their children. In that case, the rating should be read as s_i/w_i, i.e., s_i corresponds to the rating scaled by w_i.

Upon receiving ratings of the form (s_i, w_i) from all children, each intermediate node $f \in F$ can compute a weighted average of its children's ratings and its own. We give f the ability to replace the weight of child i with weight $\hat{w}_i$ if desired. Let r_f ($\hat{w}_f$) denote f's own rating (weight) that it wants to assign to itself (resp.). Its combined rating (s_f, w_f) is computed as $w_f = \hat{w}_f + \sum_i \hat{w}_i$ and $s_f = r_f \cdot \hat{w}_f + \sum_i (s_i \cdot \hat{w}_i/w_i)$. Note that care must be taken during evaluation as some of the w_i's may be 0.

f can also have its weight be a function of the weights of its children. In particular, $\hat{w}_f$ can be computed as $k_1 \sum_i w_i + k_2$, where $k_1, k_2 \geq 0$ are constants. When $k_1 = 1$ and $k_2 = 0$, for example, f's weight is equal to that of its children. f can also set $k_1 = 0$ and k_2 to a constant to have a weight independent of the weights of its children. Given this, its rating computation becomes $w_f = \sum_i w_i + \hat{w}_f$ and $s_f = \sum_i s_i + r_f \cdot \hat{w}_f$. Note that if the children are not reweighted no division is used which is more attractive for use in secure computation.

After each $f \in F$ computes its weighted rating, it returns (s_f, w_f) to R. R can now reweight all of its children using weights $\hat{w}_f$ (as described above with the exception of having no own rating or weight) and obtain the final weighted rating (s_R, w_R), where s_R/w_R is the learned rating.

B. Security Model

While the description above specifies the desired computation, a privacy-preserving solution cannot leak private information to the participants. That is, R announces I and the query parameters (e.g., the depth of the search, range of ratings, etc.) and learns the final result. No other information should be leaked to R or the other parties. We assume that the participants are semi-honest, i.e., they follow the protocol, but attempt to learn additional information if possible. This model is sensible in our setting since users are connected through social ties, and thus a trust relationships often exists. We note, however, that many building blocks used were designed to be secure in a stronger, fully malicious, model and thus additional techniques can make them robust against misbehaving parties if necessary. Section VII expands upon these issues.

We assume that two users have a (direct) communication channel only if they are directly connected in the network (i.e., they are friends). As is common (e.g., on Facebook) we assume the network topology (or user connections) are publicly available and therefore not sensitive. The inability of some participants to communicate directly requires them to pass their communication through other parties in the computation, appropriately secured (e.g., by using Public Key Infrastructure). Due to the hierarchical nature of social networks, some users might assume more active roles than others, e.g., the root and its children are more central to query execution than users farther from the root. The security guarantees are, however, required to hold for all participants, regardless of their role.

Let $P_0, P_1, \ldots, P_{m-1}$ denote the set of participating users, where $P_0 = R$. We formally define security using the standard definition in secure multi-party computation for semi-honest adversaries. Because most participants receive no output, we denote "no data" by the special character $\perp$.

Definition 1: Let $P_0, \ldots, P_{m-1}$ engage in a protocol π that computes function $f(\mathsf{in}_0, \ldots, \mathsf{in}_{m-1}) = (\mathsf{out}_0, \perp, \ldots, \perp)$, where in_i denotes the input of party P_i and out_0 is the output of party P_0. Let $\text{VIEW}_\pi(P_i)$ denote the view of participant P_i during the execution of protocol π. More precisely, P_i's view is formed by its input and any internal random coin tosses r_i, as well as messages $m_1, \ldots, m_t$ passed between the parties during protocol execution, as

$$\text{VIEW}_\pi(P_i) = (\mathsf{in}_i, r_i, m_1, \ldots, m_t).$$

We say that protocol π is secure against semi-honest adversaries if for each party P_i there exists a probabilistic polynomial time simulator S_i such that

$$\{S_i(f(\mathsf{in}_0, \ldots, \mathsf{in}_{m-1}))\} \equiv \{\text{VIEW}_\pi(P_i), \mathsf{out}_i\},$$

where $\mathsf{out}_i = \perp$ for all parties except P_0 and $\equiv$ denotes computational indistinguishability (using an appropriate security parameter).

Note that this standard model allows participants to collude (i.e., share the information). The security guarantees must hold as long as the coalition size does not exceed a threshold t. In particular, this means that the root R does not have exceptional capabilities and thus participates as a regular user.

C. Overview of Techniques

To carry out the computations securely, we employ a semantically secure public-key homomorphic encryption scheme, i.e., one which maps operations on ciphertexts to corresponding operations on the underlying plaintexts. In particular, we employ additively homomorphic encryption with the following properties: given messages m_1, m_2 and encryption algorithm Enc, $\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 + m_2)$. This implies $\mathsf{Enc}(m_1)^c = \mathsf{Enc}(c \cdot m_1)$ for $c > 0$. We assume an encryption scheme $\mathcal{E}$ is composed of three algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for key generation, encryption, and decryption, respectively.

To perform our protocol we must add two unknown values, multiply an unknown quantity by a known value, and divide

two encrypted numbers. The first two operations are easily accomplished via homomorphic encryption, while division requires additional techniques. For this reason we develop a division protocol that takes two encrypted inputs and produces an encrypted quotient. Custom protocols for secure division exist in the literature (e.g., [1]–[3]), but none of can be used in our setting. More details are given in Section III-C.

One complication of developing division protocols is handling division by zero. Since in our scheme the child's weight can be zero, our solution must not fail in such cases. Consequently in our division protocol division by zero returns the largest possible value. To ensure the overall computation is performed correctly when reweighting the child, we therefore compute the rating given (s_f, w_f) as $w_f = \hat{w}_f + \sum_i (\hat{w}_i \cdot b_i)$ and $s_f = r_f \cdot \hat{w}_f + \sum_i (s_i \cdot \hat{w}_i \cdot b_i / w_i)$, where b_i is a bit which is set to 1 iff w_i is non-zero. This means that we also have a procedure (in Section III-D) for testing whether a ciphertext corresponds to an encryption of a non-zero value, which outputs an encryption of a bit.

To ensure that no single party or coalition of less than t users can recover any values, we employ a (t, n)-threshold encryption scheme, where n users receive shares of the decryption key and participation of at least t users is required to decrypt any value. Any coalition of less than t users learns nothing. Since no single party is assumed to be fully trusted, we also require a fully distributed key generation protocol. In particular, neither the root, nor the system should be able to decrypt user's inputs, and therefore neither can be used as a trusted "dealer" for key generation. Considering these requirements, our implementation uses Paillier encryption.

One of the major challenges in designing privacy-preserving protocols stems from the inefficiency of existing techniques. While complexity is often measured in terms of communication and computation, in our setting it is especially important to minimize interaction as measured in the number of rounds. This is due to the fact that every round of interaction requires every user to pass information through the social networking site in order to proceed. As we would like to minimize the latency, minimizing the number of rounds is an important goal.

Once a public key for a threshold homomorphic scheme is generated, all users at the bottom of the hierarchy can produce their ratings and forward them to their parents without any further interactions. Similarly, intermediate nodes can independently compute a combined rating from the encrypted values received from their children (we assume intermediate nodes follow the second type of computation described in Section II-A). If reweighting is desirable, interactive division protocols must be performed. To achieve reasonable efficiency, we reserve the ability to reweight children's ratings for the root node, where such computation can make the most significant difference. In the description of our protocol (Section IV) we assume that intermediate nodes do not reweight ratings obtained from their children.

III. Background and Building Blocks

A. Threshold Paillier Encryption Scheme

The Paillier cryptosystem is a semantically secure, additively homomorphic, public-key encryption scheme based on the composite residuosity problem [4]. It was expanded in [5], [6] to function as a threshold encryption scheme, but required the use of a trusted dealer to distribute the keys to participants. The reliance on a trusted dealer was lifted in [7], [8]. We thus obtain a fully distributed scheme composed of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, which we briefly sketch next. Our current description covers only material necessary for understanding this work; a detailed description is given in [9].

$\mathsf{Gen}(\kappa, t, n)$ is a protocol run between n parties to setup a (t, n)-threshold scheme. The parties first generate n additive shares of two $\kappa/2$-bit candidate primes p and q, compute $N = pq$, and test whether N is indeed a product of two large primes. The parties set the public key pk to (N, g), where $g = N + 1$, then compute the private key $sk = d$, where each party obtains a (t, n)-share of d.

$\mathsf{Enc}(pk, m)$ is a probabilistic algorithm which, on input message $m \in \mathbb{Z}_N^*$ and public-key pk, outputs ciphertext $c \in \mathbb{Z}_{N^2}$, using randomness $r \xleftarrow{R} \mathbb{Z}_N$.

$\mathsf{Dec}(pk, d_1, \ldots, d_k, c)$ is a protocol run by $k \geq t$ parties where each party first computes c_i from ciphertext c and its share d_i of the decryption key. Each party then combines any t c_i's to recover the plaintext m.

To compute N during the distributed key generation phase the primality of p and q is tested concurrently. Thus if finding a $\kappa/2$-bit prime takes on the order of $\kappa/2$ trials (and thus κ trials for both), the distributed computation requires $\kappa^2/4$ trials.

B. Supplemental Operations

Re-randomization. Given a ciphertext $\mathsf{Enc}(v)$, we will need to re-randomize it so there is no correlation between the original and new ciphertexts without affecting the underlying plaintext. Using homomorphic encryption, this is easily accomplished by multiplying the ciphertext by $\mathsf{Enc}(0)$.

Additive sharing. We also need to additively split a value modulo N among n participants. Given $\mathsf{Enc}(v)$, each party i for $i = 0, \ldots, n-2$ chooses $r_i \xleftarrow{R} \mathbb{Z}_N$, encrypts, and broadcasts $\mathsf{Enc}(r_i)$. Given $\mathsf{Enc}(v), \mathsf{Enc}(r_0), \ldots, \mathsf{Enc}(r_{n-2})$, each P_i locally computes $\mathsf{Enc}(r_{n-1}) = \mathsf{Enc}(v - \sum_{i=0}^{n-2} r_i) = \mathsf{Enc}(v) \prod_{i=0}^{n-2} \mathsf{Enc}(r_i)^{-1}$. All parties then jointly decrypt $\mathsf{Enc}(r_{n-1})$ for P_{n-1} (i.e., only P_n learns r_{n-1}). We use $[v]$ to denote that v is additively split among the n users mod N.

Multiplication. Our solution relies on the ability to securely multiply two values. When both operands to the protocol are encrypted, the protocol, Mult, proceeds as follows:

$\mathsf{Mult}(\mathsf{Enc}(v_1), \mathsf{Enc}(v_2))$:

1) The parties additively split and decrypt shares of the second argument to jointly hold $[v_2]$ (i.e., only party i learns ith share, as described above).

2) Each party i computes $\mathsf{Enc}(u^{(i)}) = \mathsf{Enc}(v_2^{(i)} \cdot v_1) = \mathsf{Enc}(v_1)^{v_2^{(i)}}$, where $v^{(i)}$ denotes the share of v that party i has, then re-randomizes the result, and broadcasts it.
3) Each party locally assembles $\mathsf{Enc}(u) = \mathsf{Enc}(v_1 \cdot v_2)$ using the broadcasted values as $\mathsf{Enc}(u) = \mathsf{Enc}(\sum_{i=1}^{n} u^{(i)}) = \prod_{i=1}^{n} \mathsf{Enc}(u^{(i)})$.

During our protocol there are instances where one of the values is already in additively split form. As such the first step can be skipped, and we denote this variant as $\mathsf{Mult}(\mathsf{Enc}(v_1), [v_2])$.

C. Division Protocol

As previously mentioned, our computation requires carrying out the division operation where both operands and the output must remain secure. Division protocols in secure multi-party setting previously appeared in the literature: Atallah et al. [1] gives a two-party solutions based on the Newton method that uses homomorphic encryption. Bunn and Ostrovsky [2] give a conventional implementation of division also based on homomorphic encryption for the two-party case. The latter protocol was extended by Blanton and Aliasgari [3] to the multi-party case, but was based on number-theoretic techniques (i.e., a linear secret sharing scheme). In this work we start with a lightweight version of Bunn-Ostrovsky's solution and extend it to work with multiple parties.

Suppose we wish to divide some integer v by another integer d. The protocol of Bunn-Ostrovsky computes each bit of the quotient $q = \lfloor v/d \rfloor$ securely. Let $v = v_{\ell-1} \cdots v_0$ denote the binary representation of ℓ-bits of v, where v_0 is the least significant bit. The algorithm first sets the remainder to be v and tests whether subtracting $2^{\ell-i}d$ results in a negative value, starting from $i = 1$. If the remainder is greater or equal to $2^{\ell-i}d$, the $(\ell - i)$th bit of the result is set to 1, the remainder is decremented by $2^{\ell-i}d$ (otherwise, the remainder is unchanged and the $(\ell - i)$th bit of the result is 0), and i is incremented. The main inefficiency of this protocol is the need to determine for what value of i the quantity $2^i d$ exceeds the modulus N. This causes the protocol to take at least $O(|N|)$ rounds, where the amount of communication is also large. (Note that a protocol using linear secret sharing is much more efficient in this respect, as the size of the modulus can be chosen to be very close to the size of values.) When, however, the size of the modulus $|N|$ is significantly larger than the size of integers on which we operate, the protocol can be specified to use values of i in a much smaller range so that $2^i d$ will never exceed the modulus. Thus, assuming that the size of the modulus $|N|$ is larger than twice the size of the maximum values of v and d, we significantly decrease the complexity of the protocol by skipping $O(|N|)$ checks.

This solution requires both operands be shared encrypted in a bitwise manner. As such, we employ the protocol of Schoenmakers and Tuyls [10] which, given an encrypted value, allows the parties to generate encryptions of its bitwise representation (this conversion is also necessary in the Bunn-Ostrovsky solution, but was not described). This protocol, denoted Bits, can be specified to compute any number ℓ of the least significant bits of the encrypted value, in which case its complexity is linear in ℓ. In what follows, we will explicitly specify the number of bits to output as a parameter to this protocol, i.e., we use $\langle \mathsf{Enc}(v_{\ell-1}), \ldots, \mathsf{Enc}(v_0) \rangle \leftarrow \mathsf{Bits}(\ell, \mathsf{Enc}(v))$.

The division protocol of [2] also uses a sub-protocol (denoted MinLoc) which, given two values, outputs the location of the minimum value as a bit, i.e., on input (v_1, v_2), it outputs 0 iff $v_1 \leq v_2$. The arguments are assumed to be additively shared (modulo $|N|$) in bitwise form. (For more implementation details see [2].) In our solution, we use a similar building block that instead computes "less than or equal," i.e., on input v_1, v_2 (in bitwise additively split form), LessOrEqual outputs 1 iff $v_1 \leq v_2$ (the complement of MinLoc). This optimization decreases the number of calls to Bits (one of the most expensive building blocks) by a factor of 2.

Another optimization employed reduces the size of the numbers on which LessOrEqual is called from 2ℓ to $\ell + 1$. Namely, to subtract $2^{\ell-i}d$ from the current remainder, both the value and the remainder are represented as 2ℓ-bit values ($2^{\ell-i}d$ is formed by prepending it with i zero bits, and setting the ℓ most significant bits to 0). Note that the result remains the same if the ℓ most significant bits of $2^{\ell-i}d$ are combined into a single bit. Thus we set the $\ell+1$st bit of $2^{\ell-i}d$ to be 1 if one or more of the ℓ most significant bits of its 2ℓ-bit representation are set. These $\ell + 1$ bits can be precomputed for all i's once, and LessOrEqual can be called on the $\ell + 1$ bit values. We also convert d to additive shares to avoid performing multiple calls inside LessOrEqual (for use in Mult).

$\mathsf{Div}(\mathsf{Enc}(v), \mathsf{Enc}(d))$:

1) The n parties execute $\langle \mathsf{Enc}(d_{\ell-1}), \ldots, \mathsf{Enc}(d_0) \rangle \leftarrow \mathsf{Bits}(\ell, d)$, where ℓ is the maximum length of plaintext integers, and then additively share each of (encrypted) d_i and jointly decrypt each share for the intended recipient.
2) They compute the $\ell+1$st bit of $2^i d$, b_i, for $i = 1, \ldots, \ell - 2$ by setting $\mathsf{Enc}(b_1) = \mathsf{Enc}(d_{\ell-1})$ and computing $\mathsf{Enc}(b_{i+1}) = \mathsf{Enc}(\mathsf{OR}(b_i, d_{\ell-i})) = \mathsf{Enc}(b_i) \cdot \mathsf{Enc}(d_{\ell-i}) \cdot (\mathsf{Mult}(\mathsf{Enc}(b_i), [d_{\ell-i}]))^{-1} = \mathsf{Enc}(b_i + d_{\ell-i} - b_i \cdot d_{\ell-i})$.
3) The parties set $\mathsf{Enc}(R) = \mathsf{Enc}(v)$ and for $i = 1, \ldots, \ell - 1$, they compute:
   a) The parties jointly execute $\langle \mathsf{Enc}(R_{\ell-1}), \ldots, \mathsf{Enc}(R_0) \rangle \leftarrow \mathsf{Bits}(\ell, R)$ and prepend an encrypted 0 to the result to form the $(\ell + 1)$-bit representation of R.
   b) Each party computes a share of the $(\ell + 1)$-bit representation of $2^{\ell-i}d$ by shifting the shares of i most significant bits of d $\ell - i$ positions left, appending $\ell - i$ additively shared zero bits to the result, and then prepending additively shared bit $b_{\ell-i}$ (jointly computed from $\mathsf{Enc}(b_{\ell-i})$). Denote the result by $[R'_\ell], \ldots, [R'_0]$.
   c) The parties execute $[q_{\ell-i}] \leftarrow \mathsf{LessOrEqual}(\langle [R'_\ell], \ldots, [R'_0] \rangle, \langle \mathsf{Enc}(R_\ell), \ldots, \mathsf{Enc}(R_0) \rangle)$.
   d) To compute a new remainder $R = R - q_{\ell-i}R'$, each party computes its share of $[R'] = \sum_{i=1}^{\ell} 2^{\ell-i}[R'_{\ell-i}]$, then encrypts and broadcast the result. All parties locally multiply the ciphertexts to obtain $\mathsf{Enc}(R')$ and execute $\mathsf{Enc}(q_{\ell-i}R') \leftarrow \mathsf{Mult}(\mathsf{Enc}(R'), [q_{\ell-i}])$. Finally, each party locally computes $\mathsf{Enc}(R) = \mathsf{Enc}(R) \cdot$

$\mathsf{Enc}(q_{\ell-i}R')^{-1} = \mathsf{Enc}(R - q_{\ell-i}R')$.

4) Each party locally computes its share of output q as $[q] = \sum_{i=1}^{\ell} 2^{\ell-i}[q_{\ell-i}]$, then encrypts and broadcasts its share, so that $\mathsf{Enc}(q)$ can be locally computed using the homomorphic properties of encryption.

Optionally, assembly of the quotient q in the last step can be omitted if it is beneficial to have the result in a bitwise form.

D. Non-Zero Test

Given the tools described it is straightforward to implement a non-zero test to ensure that division is performed correctly.

$\mathsf{NotZero}(\mathsf{Enc}(v))$:

1) (Optional) The parties jointly execute $\langle \mathsf{Enc}(v_{\ell-1}), \ldots, \mathsf{Enc}(v_0)\rangle \leftarrow \mathsf{Bits}(\ell, v)$, additively share each encrypted v_i and decrypt the shares to obtain $[v_{\ell-1}], \ldots, [v_0]$.
2) The parties execute $[b] \leftarrow \mathsf{LessOrEqual}(\langle [v_{\ell-1}], \ldots, [v_0]\rangle, \langle [0], \ldots, [0]\rangle)$.
3) The parties set their output to $[\overline{b}] = [1] - [b]$.

The first step is optional if the encryption is already available in bitwise form, as is the case in our main protocol.

IV. Protocol Description

In describing the protocol, we use the following notation: let R be the root, $F = \{f_1, \ldots, f_{n-1}\}$ denote the friends the root selected to query, and t be the desired decryption threshold. Also, let $\mathsf{Get\text{-}Own\text{-}Rating}(item)$ and $\mathsf{Get\text{-}Own\text{-}Weight}(item)$ be functions that return the current user's rating and weight for $item$, respectively We then have:

$\mathsf{Calc\text{-}Rating}(depth, item)$:

1) R initiates the generation of a public-private key pair for a (t, n)-threshold homomorphic encryption scheme, at the end of which each f_i and R obtain shares of the private decryption key and the public key is known to everyone.
2) Each f_i separately invokes $\mathsf{Calc\text{-}Node\text{-}Rating}(depth - 1, item)$ and publishes the result $\langle \mathsf{Enc}(s_{f_i}), \mathsf{Enc}(w_{f_i})\rangle$.
3) $f_1, \ldots, f_{n-1}$ and R jointly execute $\mathsf{Enc}(q_{f_i}) \leftarrow \mathsf{Div}(\mathsf{Enc}(s_{f_i}), \mathsf{Enc}(w_{f_i}))$ and $[b_i] \leftarrow \mathsf{NotZero}(\mathsf{Enc}(w_{f_i}))$ for each $i = 1, \ldots, n-1$.
4) R chooses weights $\hat{w}_{f_1}, \ldots, \hat{w}_{f_{n-1}}$ that it wishes to assign to friends $f_1, \ldots, f_{n-1}$, respectively, computes $\mathsf{Enc}(w) = \mathsf{Enc}(\sum_{i=1}^{n-1} b_i \cdot \hat{w}_{f_i}) = \prod_{i=1}^{n-1} \mathsf{Enc}(b_i)^{\hat{w}_{f_i}}$ and $\mathsf{Enc}(s'_{f_i}) = \mathsf{Enc}(\hat{w}_{f_i} \cdot q_{f_i}) = \mathsf{Enc}(q_{f_i})^{\hat{w}_{f_i}}$ for $i = 1, \ldots, n-1$. R re-randomizes each computed ciphertext $\langle \mathsf{Enc}(w), \mathsf{Enc}(s'_{f_1}), \ldots, \mathsf{Enc}(s'_{f_{n-1}})\rangle$ and publishes them.
5) $f_1, \ldots, f_{n-1}$ and R jointly compute $\mathsf{Enc}(s''_{f_i}) \leftarrow \mathsf{Mult}(\mathsf{Enc}(s'_{f_i}), [b_i])$ and then each participant locally computes $\mathsf{Enc}(s) = \mathsf{Enc}(\sum_{i=1}^{n-1} s''_{f_i}) = \prod_{i=1}^{n-1} \mathsf{Enc}(s''_{f_i})$.
6) $f_1, \ldots, f_{n-1}$ and R compute $\mathsf{Enc}(a) \leftarrow \mathsf{Div}(\mathsf{Enc}(s), \mathsf{Enc}(w))$ and decrypt $\mathsf{Enc}(a)$ for R, who learns a.

$\mathsf{Calc\text{-}Node\text{-}Rating}(depth, item)$:

1) If $depth \neq 0$, choose a subset of friends F' to query and forward them the request. Each friend $f_j \in F'$ executes $\mathsf{Calc\text{-}Node\text{-}Rating}(depth - 1, item)$ and forwards the result $(\mathsf{Enc}(s_{f_j}), \mathsf{Enc}(w_{f_j})$ to the requester.
2) Set encryption of own rating to be $\mathsf{Enc}(w') = \mathsf{Enc}(\mathsf{Calc\text{-}Own\text{-}Weight}(item))$, $\mathsf{Enc}(s') = \mathsf{Enc}(w \cdot \mathsf{Get\text{-}Own\text{-}Rating}(item))$.
3) If $depth \neq 0$, compute $\mathsf{Enc}(s) = \mathsf{Enc}(s' + \sum_{f_j \in F'} s_{f_i}) = \mathsf{Enc}(s) \cdot \prod_{f_j \in F'} \mathsf{Enc}(s_{f_j})$ and $\mathsf{Enc}(w) = \mathsf{Enc}(w' + \sum_{f_j \in F'} w_{f_j}) = \mathsf{Enc}(w') \cdot \prod_{f_j \in F'} \mathsf{Enc}(w_{f_j})$; otherwise set $\mathsf{Enc}(s) = \mathsf{Enc}(s')$ and $\mathsf{Enc}(w) = \mathsf{Enc}(w')$.
4) Return $\langle \mathsf{Enc}(s), \mathsf{Enc}(w)\rangle$.

Note that we allow the depth to be arbitrary. Due to the small world hypothesis, however, we advise a depth of 2, as even then a large number of users are likely to be queried.

A. Analysis

To show the security of our solution, we must show that any $t - 1$ colluding participants cannot recover more information than what is obtainable if a trusted third party implemented the functionality. In our analysis we simulate the view of the root and $t - 2$ other participants (since the root is the only participant who learns any output) thereby using the strongest adversary in the simulation. As our solution relies on several existing building blocks which are provably secure (e.g., bit decomposition), we assume their security. Since our simulation argument is standard, we provide only an outline.

In general it is trivial to simulate an encrypted value in the presence of $t - 1$ participants by encrypting any value, since the encryption scheme is semantically secure. Similarly, an additively split value can be simulated by choosing $t-1$ shares at random, and such shares will be identically distributed to real shares. Using these observations the multiplication protocol, for instance, can be fully simulated without access to the underlying data. In the division protocol the parties make calls to the bit decomposition protocol, Bits, and comparison protocol, LessOrEqual, which we assume are secure. All other computation proceeds on encrypted or additively shared values which are easily simulatable. The same applies to NotZero.

Finally, the main protocol itself is composed of the distributed key generation protocol (which has its own security proof), and the rest of the computation proceeds on encrypted values in a straightforward manner (involving Div and Mult). Therefore, at no point of can the colluding parties discover information about the intermediate values of the computation.

V. Implementation

In order to implement our protocol, we built on the Facebook social networking site. Currently Facebook offers two ways of deploying applications: web-based and desktop based.

With the web-based applications, the application interacts with Facebook using one of many web centric programming languages (e.g., Javascript, PHP) and is constrained to running inside a web browser. This is suboptimal for our purposes because (i) web browsers are restrictive environments which do not easily allow for the type of communication our protocol requires and (ii) our protocol requires large number support which these programming languages lack.

With the desktop application option, applications can be written in any language and interact with Facebook through a

common API. This API gives access to not any information available to a user logged onto Facebook. As we only require knowledge of a user's friends (i.e., the social network), this allows us to choose a programming language which has better characteristics for our problem. A desktop application has the added advantage of giving the user more control over their data (i.e., all data is stored locally), and allows the application to be run in the background unobtrusively from the user's point of view. When considering the state of currently available Facebook SDKs, and the aforementioned criteria, we chose to implement our protocol in Java.

Given these decisions, the architecture for our application is as follows. Each user first logs into Facebook using its Facebook username and password. Once the user is authenticated, it registers as online with a server. This server acts as an intermediary between all users, passing communication through the network. It also acts as a check pointing system, allowing the computation to be performed asynchronously as users log on and off the system. Once logged onto the server, the user can receive requests and issue queries. We note that our system is not dependent upon Facebook for anything other than the social network. That is, once the user logs into Facebook and the social network is retrieved, the user can immediately log out of Facebook and never log into it through the application again, instead she would rely on our application to track users. We also note that it is easy to see that our system can be used in the absence of Facebook when other social networks are available. An advantage of using Facebook, however, is that it is a widely used social networking site which allows us to message any user with a request to participate in the computation, even if the user does not have the application installed. Thus, if users are not directly involved in the computations, they can merely return their value and not install any application at all.

In order to implement the protocol, each of the building blocks described in Section III had to be implemented. To gain a better understanding of the performance of each of these operations, we present the timing results for each building block. As the computations were distributed, we simulated each user (and the server) as its own PC on a 100Mb LAN. The computers used were Sun workstations using 2200 MHz AMD processors and 2 GB of memory. We tested each building block independently, and took an average time over 10 runs, with the exception of key generation tests which were averaged over 50 runs due to large fluctuations in the number of trials. We are unaware of other implementations of secure multiparty techniques of similar complexity and scale.

A. Key Generation

As mentioned in Section III-A, fully distributed generation of an RSA modulus normally involves first choosing candidates for primes p and q and testing them simultaneously, resulting in an expected $O(\kappa^2)$ trials for a κ-bit modulus [7]. As $O(\kappa^2)$ trials is computationally infeasible, we applied the optimization techniques of [11] which allow for significant performance improvements over the Boneh-Franklin biprimality test while keeping each test efficient. In particular, the number of primality tests in practice reduces to about 1000 for a 1024-bit modulus. Our implementation followed the description given in [11] with the exception that threading was not included. Our results are consistent with those reported in [11]. Additionally while the experiments in [11] included at most 5 parties, we show performance of the algorithm up to 19 participants. We also suggest a slight efficiency improvement to the techniques of [11] described in Appendix A.

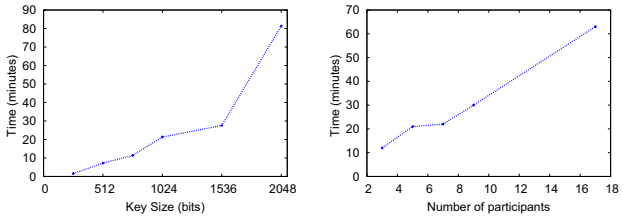

Fig. 1. Timing results for distributed key generation.

The remaining key material is computed using the techniques of [6], which also rely on results from [12]. The details are given in [9]. To the best of our knowledge, this is the first implementation of fully distributed Paillier encryption scheme.

Figure 1 illustrates performance of the distributed key generation protocol. For all experiments, we plot the performance of a protocol by varying (i) the size of the encryption key and (ii) the number of participants. In all figures, the plot on the left hand side fix the number of participants to 7 (varying key size) and the plot on the right fixes the key size to 1024 bits (varying number of participants). (In [9] we also plot the number of trials.) The results indicate that this (one-time) operation can be executed within an acceptable time frame.

There are other techniques for distributed key generation available in the literature. For example, Ibrahim [13] reduces the number of rounds from quadratic to linear; however, each round becomes expensive and is equivalent to a linear number of tests. Also, Biehl and Takagi [14] have a technique using quadratic fields which provides a possible area for improvement. We, however, anticipate that the implemented techniques would be comparable or even outperform other solutions including the newest techniques [15].

B. Protocol Performance

We now show the performance of our protocol and its components beginning with the smallest non-trivial protocol, Mult, and continuing with the remaining protocols. In addition to gaining an understanding of the most resource-intensive parts of the solution, this also gives the reader performance insights of several recent constructs with no prior implementations.

Multiplication. The results of running the multiplication protocol (Section III-B) are depicted in Figure 2. Since this protocol involves at least one modulo exponentiation, the expected function growth is cubic in the size of the key which the figure confirms. For a key size 1024, performance stays

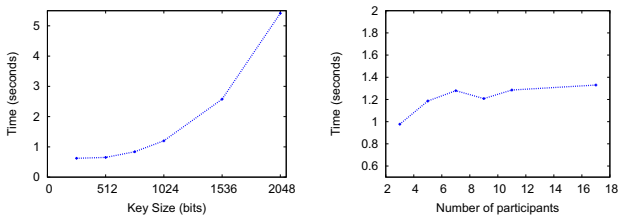

Fig. 2. Timing results for distributed multiplication operation.

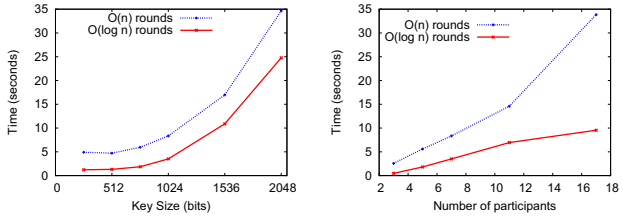

Fig. 3. Timing results for distributed random bit generation.

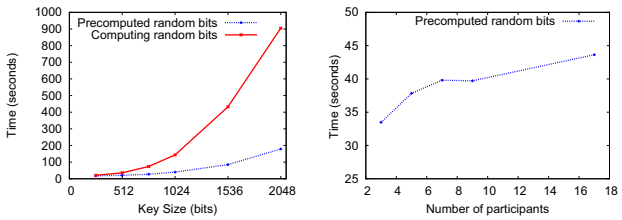

Fig. 4. Timing results for distributed bit decomposition.

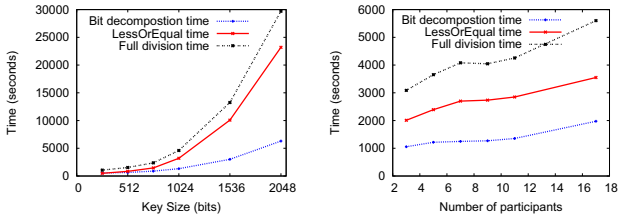

Fig. 5. Timing results for distributed division.

around 1.3 seconds even if the number of parties varies as all operations are performed by the participants in parallel.

Random bit generation. This operation allows parties to jointly choose a random bit and can be realized in linear (in the number of participants), logarithmic, or constant rounds. We implemented the first two (due to the small number of parties). Figure 3 shows the performance of both versions of the protocol. The gap between their performance grows as the number of participants increases (both require $n-1$ multiplications, but one offers more parallelism). Small fluctuations in the runtime are an artifact of the computing environment.

Bit decomposition. The bit decomposition protocol Bits [10] repeatedly generates random bits and also performs addition of integers in bitwise form. To add $a = a_{\ell-1}\ldots a_0$ and $b = b_{\ell-1}\ldots b_0$, bits a_i's are given in the clear, while bits b_i's as well as carry c_i and sum s_i bits are encrypted as $\mathsf{Enc}(b_i)$, $\mathsf{Enc}(c_i)$, and $\mathsf{Enc}(s_i)$. We use the following computation for the addition, which results in only one interactive multiplication (i.e., $\mathsf{Mult}(\mathsf{Enc}(b_i), \mathsf{Enc}(c_i))$) per bit of the computation:

$$c_{i+1} = \begin{cases} b_i \cdot c_i & \text{if } a_i = 0 \\ b_i + c_i - b_i \cdot c_i & \text{if } a_i = 1 \end{cases}$$

where $s_i = a_i + b_i + c_i - 2a_ib_i - 2a_ic_i - 2b_ic_i + 4a_ib_ic_i$. Figure 4 depicts the performance of bit decomposition for 32-bit integers with and without the overhead for random bit generation. Note the time saved by precomputing random bits. An interesting aside is the $O(\log n)$-round random bit generation in the protocol does not result in performance improvement over the $O(n)$-round version due to the high degree of parallelism introduced, i.e., Bits runs many (32 here) instances of the bit generation protocol in parallel. We plot the Bits protocol using the $O(n)$-round random bit generation.

Division. Finally Figure 5 reports the performance of division of 32-bit integers with precomputed random bits, where the time to execute Bits (33 invocations), and LessOrEqual (32 invocations on 33-bit integers), are also shown. Note the timings for Bits and LessOrEqual dominate the timing of division. For completeness, LessOrEqual is described in Appendix B.

The performance of the protocol for computing the weighted average of ratings will be dominated by division. During execution of the system, the interaction proceeds as follows:

1) The root broadcasts a request to initiate a recommendation protocol to m friends.
2) The root generates a public-private key pair with the first $n-1$ friends who respond as willing to participate; the public key is then broadcast to the remaining friends.
3) The n parties generate a sufficient number of random bits while all participating friends recursively collect and assemble their recommendations.
4) The n parties run all division protocols in parallel (m instances in round 1 and one instance in round 2).

On the scale of this type of interaction, where collecting ratings can take days, the protocol is reasonably efficient.

VI. Related Work

Prior research on related recommendation systems has two main focuses: trust-based and systems that preserve privacy of user preferences. In trust-based recommendation systems, trust is modeled using a graph where each edge weight represents the trust between the two parties it connects. One example is TrustWalker [16], where ratings of items are determined by a trust network and an item similarity metric. Another, more similar, example is FilmTrust [17], in which users specify trust values for each user (similar to our c_i) and calculate the rating of a movie as the weighted average of each user's rating, where the trust values act as weights. FilmTrust differs from our scheme in that the depth of the query can only be one, decreasing the coverage of items (i.e., if none of the root's friends have a rating, no rating is given), and there is

no privacy of user ratings. There is also other literature on trust-based collaborative recommendation systems, e.g., [18]–[20] that use trust-based mechanisms (including social trust) to improve the correctness of public recommender systems. That is, they attempt to remove the bias that could be introduced by malicious users through injection attacks via anonymous ratings. They do not, however, preserve the privacy of individual ratings.

Publications that build recommendation systems with user privacy in mind include a secure protocol by Katzenbeisser and Petković [21] for computing the similarity of vectors for use in medical applications. That is, given a vector (e.g., disease vector), the system is able to securely find similar vectors and make recommendations about doctors. Similarly, collaborative filtering techniques, where a user has a set of preferences and the system attempts to suggest related products based on similarities with other users, have been developed that take user privacy into account. Such techniques are known based on both homomorphic encryption [22], [23] and data perturbation [24]–[27]. Since collaborative filtering is used, results are obtained which are based on similarity to existing users. Thus while both recommendation systems are similar to what we present here (with the schemes based on homomorphic encryption being closer), they do not take into account trust relationships as our scheme does. This is important to note, because without the ability to use the information present in the social network, these techniques cannot be adapted to achieve the same results as our technique. Our solution therefore could allow the user to obtain different recommendations, thereby offering a potentially better overall recommendation.

From the literature on data privacy in social networks, publications in the area of access control are the closest to this work. In particular, articles such as [28]–[32] allow users to specify access rules for their content stored on the social networking site and/or preserve privacy of their data from the networking site itself. The mechanisms for achieving the goals differ, ranging from using trusted third parties to decentralized solutions where the user herself implements access control policies for her data (e.g., using public-key [30] or key management [32] techniques). The goal of these publications is, however, access control, and thus does not solve the same problem as our work, as evidenced by the fact that they reveal data which we require be kept private.

Another recent notion of privacy which is also used in recommendation systems is called "differential privacy." Loosely speaking, differential privacy [33], [34] for statistical databases guarantees that two data sets from the same population which differ by one element will occur with almost the same probability. Intuitively, this means that for each element in the database there exist very similar elements. Differential privacy is usually achieved by adding noise to the dataset, and the amount and type of noise is heavily dependent on the statistical queries that users are allowed to execute on the database. This means that there is a trade-off between accuracy and privacy. In application to recommendation systems, McSherry and Mironov [35] built a system to achieve differential privacy over the Netflix dataset. Experiments on their system showed that the accuracy of results was not severely effected throughout the query execution by maintaining privacy of user data.

Another work by Kearns et al. [36] develops secure computation techniques for a network of users. That work presents a mechanism for securely computing belief propagation and Gibbs sampling, which is not applicable to the computation we perform in our application. The authors assume a fully decentralized network, which can also be easily achieved in our setting (i.e., our implementation used a centralized server for the ease of synchronizing the interaction, but this is not an intrinsic requirement of the solution).

VII. Discussion and Conclusions

In this work we design and implement a protocol that uses social network connections to realize a reliable recommendation system that preserves the privacy of individual user ratings. We assumed semi-honest participant behavior, justified by social ties. It is important to note, however, that additional techniques can be employed to prevent user misbehavior. In what follows, we describe a number of such techniques including countermeasures that are not covered by, and thus go beyond, the techniques for secure multi-party computation in the presence of active adversaries.

One concern that potential users might have is an uncertainty about whether their privacy is sufficiently protected through the participation of enough individuals. As mentioned previously, each user can decline to participate if the threshold t is set below her tolerance level. Furthermore, because all (secured) communication is relayed through the root, the user might want to verify that the encryption key was generated honestly (as opposed to by e.g., users created by the root) and there are a certain number of other users contributing their inputs. To alleviate such concerns, all communication relayed through the root can be authenticated, e.g., signed by the originating parties using a Public Key Infrastructure (PKI) embedded in the social network. This prevents all parties (not just the root) from faking additional users or pretending to be a different user. Furthermore, when a friend of the root sends a message to its children, the authentication information associated with the users who participated in the creation of the encryption key can also be passed to them. This will allow all participating users (rather than just the root's friends) to verify that their data will be adequately protected.

If due to some reason a user still does not feel that her data is being adequately protected even in the presence of the above safeguards, the user can decline to participate. For example, a malicious querier can run multiple queries on the same product selectively excluding users in the attempt to learn individual responses. This concern is not unique to our setting, but exists every time the flexibility of the system allows for overlapping queries. Since the querier and the product being queried are known, however, a user can easily detect such queries and refuse to participate in the computation on all consecutive query runs. In this way topological attacks against

the system can be mitigated. The user also has an oblivious way of declining to participate by submitting a 0 rating (this is especially meaningful in social networks where users may not want to offend their friends by declining to participate). It is important to note that the computation of the result requires at least t users to participate. This means that performing the computation multiple times (i.e., one time including the information given by user U, and another not) is impossible without the collusion of t users. Since we assume this is not the case, users are safe from such attacks.

We note that the aggregation strategy used by this protocol is easily modifiable at the user level. When a user is asked to return a rating, they are not required to incorporate the knowledge of their neighborhood. Similarly, they are not required to incorporate their personal knowledge. Each individual user is instead allowed to amalgamate the results in the way they deem most accurately reflects their rating for the item. This is very powerful as it allows users to easily and quickly alter how their ratings are generated (e.g., via a web interface) without changing the underlying protocol.

While in the above we focused on the protection of the users contributing their ratings, a concern the querier might have is that a single invalid user rating can render the entire computation useless. For example, a user can submit a rating significantly exceeding the maximum valid rating or a negative rating. This problem can be addressed at relatively low cost by enforcing range checking on each rating: For a leaf node, such verification can be performed using range or set membership zero-knowledge proof of knowledge of their rating and weight, which are known for Paillier encryption. For intermediate nodes that do not have knowledge of a rating they assemble in encrypted form, range checks instead can be performed using joint execution of LessOrEqual on the value s_i/w_i. While a user can always set her rating to any value in the range, its effect on the overall result will be marginal as it will be averaged with all other friends' ratings based on a selected weight.

Finally, we note that our implementation of the comparison operation is based on homomorphic encryption, while recent literature [37] reports that for the two-party case optimized implementations of generic garbled circuits [37]–[39] provide the best performance results. There are no equivalent studies for the multi-party case, but the efficiency of alternative implementations is worth investigating in the future.

VIII. Acknowledgments

Research was sponsored in part by the grant AFOSR-FA9550-09-1-0223, and the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara, "Private collaborative forecasting and benchmarking," in *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2004, pp. 103–114.

[2] P. Bunn and R. Ostrovsky, "Secure two-party k-means clustering," in *ACM Conference on Computer and Communications Security (CCS)*, 2007, pp. 486–497.

[3] M. Blanton and M. Aliasgari, "Secure computation of biometric matching," Department of Computer Science and Engineering, University of Notre Dame, Tech. Rep. 2009–03, 2009.

[4] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology – EUROCRYPT*, ser. LNCS, vol. 1592, 1999, pp. 223–238.

[5] P.-A. Fouque, G. Poupard, and J. Stern, "Sharing decryption in the context of voting or lotteries," in *International Conference on Financial Cryptography (FC)*, ser. LNCS, vol. 1962, 2000, pp. 90–104.

[6] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of Paillier's probabilistic public-key system," in *International Workshop on Practice and Theory in Public Key Cryptography (PKC)*, 2001, pp. 119–136.

[7] D. Boneh and M. Franklin, "Efficient generation of shared RSA keys," *Journal of the ACM*, vol. 48, no. 4, pp. 702–722, 2001.

[8] I. Damgård and M. Koprowski, "Practical threshold RSA signatures without a trusted dealer," in *Advances in Cryptology – EUROCRYPT*, ser. LNCS, vol. 2045, 2001, pp. 152–165.

[9] T. R. Hoens, M. Blanton, and N. Chawla, "A private and reliable recommendation system using a social network," Department of Computer Science and Engineering, University of Notre Dame, Tech. Rep. 2010–05, 2010.

[10] B. Schoenmakers and P. Tuyls, "Efficient binary conversion for Paillier encrypted values," in *Advances in Cryptology – EUROCRYPT*, ser. LNCS, vol. 4004, 2006, pp. 522–537.

[11] M. Malkin, T. Wu, and D. Boneh, "Experimenting with shared generation of RSA keys," in *Symposium on Network and Distributed System Security (NDSS)*, 1999, pp. 43–56.

[12] Y. Frankel, P. MacKenzie, and M. Yung, "Robust efficient distributed RSA-key generation," in *ACM Symposium on Theory of Computing (STOC)*, 1998, pp. 663–672.

[13] M. Ibrahim, "Eliminating quadratic slowdown in two-prime RSA function sharing," *International Journal of Network Security (IJNS)*, vol. 7, no. 1, pp. 106–113, 2008.

[14] I. Biehl and T. Takagi, "A new distributed primality test for shared RSA keys using quadratic fields," in *Australasian Conference on Information Security and Privacy (ACISP)*, 2002, pp. 1–16.

[15] I. Damgård and G. Mikkelsen, "Efficient, robust and constant-round distributed RSA key generation," in *Theory of Cryptography Conference (TCC)*, 2010, pp. 183–200.

[16] M. Jamali and M. Ester, "Trustwalker: a random walk model for combining trust-based and item-based recommendation," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2009, pp. 397–406.

[17] J. Golbeck and J. Hendler, "FilmTrust: Movie recommendations using trust in web-based social networks," in *IEEE Consumer Communications and Networking Conference (CCNC)*, 2006, pp. 282–286.

[18] P. Massa and P. Avesani, "Trust-aware collaborative filtering for recommender systems," in *International Conference on Cooperative Information Systems (CoopIS)*, ser. LNCS, vol. 3290, 2004, pp. 492–508.

[19] Y.-M. Li and C.-P. Kao, "TREPPS: A trust-based recommender system for peer production services," *Expert Systems with Applications (ESWA)*, vol. 36, no. 2, pp. 3263–3277, 2009.

[20] F. Zhang, L. Bai, and F. Gao, "A user trust-based collaborative filtering recommendation algorithm," in *International Conference on Information and Communications Security (ICICS)*, 2009, pp. 411–424.

[21] S. Katzenbeisser and M. Petkovic, "Privacy-preserving recommendation systems for consumer healthcare services," in *International Conference on Availability, Reliability and Security (ARES)*, 2008, pp. 889–895.

[22] J. Canny, "Collaborative filtering with privacy," in *IEEE Symposium on Security and Privacy*, 2002, pp. 45–57.

[23] ——, "Collaborative filtering with privacy via factor analysis," in *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2002, pp. 238–245.

[24] H. Polat and W. Du, "Privacy-preserving collaborative filtering using randomized perturbation techniques," in *IEEE International Conference on Data Mining (ICDM)*, 2003, pp. 625–628.
[25] ——, "SVD-based collaborative filtering with privacy," in *ACM Symposium on Applied Computing (SAC)*, vol. 1, 2005, pp. 791–795.
[26] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar, "On the privacy preserving properties of random data perturbation techniques," in *IEEE International Conference on Data Mining (ICDM)*, 2003, pp. 99–106.
[27] S. Zhang, J. Ford, and F. Makedon, "A privacy-preserving collaborative filtering scheme with two-way communication," in *ACM Conference on Electronic Commerce (EC)*, 2006, pp. 316–323.
[28] B. Carminati and E. Ferrari, "Privacy-aware collaborative access control in web-based social networks," in *IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec)*, ser. LNCS, vol. 5094, 2008, pp. 81–96.
[29] J. Domingo-Ferrer, "A public-key protocol for social networks with private relationships," in *Modeling Decisions for Artifical Intelligence (MDAI)*, ser. LNCS, vol. 4617, 2007, pp. 373–379.
[30] M. Lucas and N. Borisov, "FlyByNight: Mitigating the privacy risks of social networking," in *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2008, pp. 1–8.
[31] J. Domingo-Ferrer, A. Viejo, F. Sebe, and U. Gonzalez-Nicolas, "Privacy homomorphisms for social networks with private relationships," *Computer Networks (CN)*, vol. 52, no. 15, pp. 3007–3016, 2008.
[32] K. Frikken and P. Srinivas, "Key allocation schemes for private social networks," in *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2009, pp. 11–20.
[33] C. Dwork, "Differential privacy," in *International Colloquium on Automata, Languages and Programming (ICALP)*, ser. LNCS, vol. 4052, 2006, pp. 1–12.
[34] ——, "Differential privacy: A survey of results," in *International Conference on Theory and Applications of Models of Computation (TAMC)*, ser. LNCS, vol. 4978, 2008, pp. 1–19.
[35] F. McSherry and I. Mironov, "Differentially private recommender systems: Building privacy into the Netflix prize contenders," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2009, pp. 627–636.
[36] M. Kearns, J. Tan, and J. Wortman, "Privacy-preserving belief propagation and sampling," in *Advances in Neural Information Processing Systems (NIPS)*, 2007.
[37] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *Cryptology and Network Security (CANS)*, 2009, pp. 1–20.
[38] A. Yao, "How to generate and exchange secrets," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1986, pp. 162–167.
[39] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *ACM Conference on Electronic Commerce (EC)*, 1999, pp. 129–139.
[40] M. Ben-Or, S. Goldwasser, and A. Widgerson, "Completeness theorems for non-cryptographic fault tolerant distributed computation," in *ACM Symposium on Theory of Computing (STOC)*, 1988, pp. 1–10.

APPENDIX A
EFFICIENCY IMPROVEMENT OF THE DISTRIBUTED RSA KEY GENERATION

In order to improve the efficiency of RSA distributed key generation, Malkin, Wu, and Boneh [11] outline several practical optimizations. The first, and most important, of which is distributed sieving (others being trial division and threading to run multiple tests in parallel).

In distributed sieving, each user chooses a random number which is coprime to the product of all small primes less than the sieving bound, M. The users can then multiply all of their shares to get $a = \prod_{i=1}^{n} a_i$, where a is also coprime to M. In order to split the product a into additive shares, the paper recommends running the BGW protocol [40] for secure circuit evaluation, which in this case performs multiplication of two values shared using a linear secret sharing scheme, once for each party. That is, the parties sequentially compute $\prod_{i=1}^{k} a_i$ for $k = 2, 3, \ldots, n$. We note, however, that instead of running the BGW protocol sequentially, one can run multiple instances in parallel. That is, the participants can run the BGW protocol to compute shares of $a_{2j-1} \cdot a_{2j}$ for each $j = 1, \ldots, \lfloor n/2 \rfloor$, resulting in $\lfloor n/2 \rfloor$ protocols executed in parallel. Once these have been computed, the users can continue multiplying pairs of shared values together, until finally the entire product has been shared using linear secret sharing between all of the parties.

This optimization results in the splitting of the multiplicative shares into additive shares taking a logarithmic number of rounds in the number of users as opposed to linear. In the case of a user with a large number of friends, this may result in a large speedup.

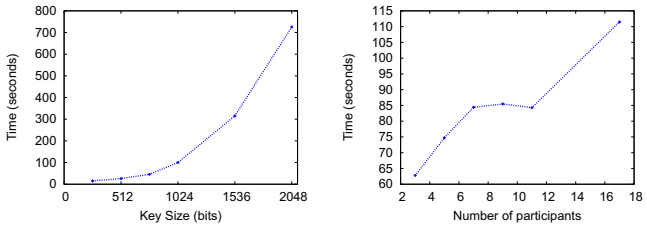

Fig. 6. Timing results for distributed comparison.

APPENDIX B
OVERVIEW OF LessOrEqual PROTOCOL AND PERFORMANCE

Our realization of LessOrEqual involves computation of location of minimum LocMin using two (unknown) arguments in bitwise form and then complementing the resulting bit. In [2], LocMin functionality was implemented using a standard minimum comparison of two values a and b in a bitwise form as described next. Let $a = a_{\ell-1} \ldots a_0$, $b = b_{\ell-1} \ldots b_0$, and let $\oplus$ denote XOR.

$$\begin{aligned}\mathsf{LocMin}(a,b) \quad = \quad & (a_0 \oplus b_0)a_0 + (a_0 \oplus b_0 \oplus 1)(a_1 \oplus b_1)a_1 + \\ & + (a_0 \oplus b_0 \oplus 1)(a_1 \oplus b_1 \oplus 1)(a_2 \oplus b_2)a_2 + \\ & + \cdots + \\ & + (a_0 \oplus b_0 \oplus 1) \cdots (a_{\ell-1} \oplus b_{\ell-1})a_{\ell-1}\end{aligned}$$

The XOR operation involves one multiplication as in $x \oplus y = x + y - 2xy$. Then the overall computation is dominated by $O(\ell)$ executions of Mult protocol in $O(\ell)$ rounds.

Our implementation of LessOrEqual takes the arguments in bitwise form, where the bits b_i of the second argument are encrypted, while having the bits of the first argument in the additively shared form allows for reduced runtime of the multiplication protocol. All additions are performed on encrypted values using the homomorphic property of the encryption scheme. Figure 6 shows performance of our implementation of LessOrEqual for $\ell = 32$ with a fixed number of participants (7) and a fixed key size (1024 bits).