Privacy-Preserving Profile Matching Using the Social Graph

Arjan Jeckmans, Qiang Tang, and Pieter Hartel
Distributed and Embedded Security, University of Twente, the Netherlands
{a.j.p.jeckmans,q.tang,pieter.hartel}@utwente.nl

Abstract

We present a privacy-preserving protocol for users to test a match with potential new friends in an environment where all users cryptographically encrypt their private information. The following scenario is considered. Suppose that user Alice thinks that Bob might be a good new friend. So, Alice and the Online Social Network (representing Bob) engage in a two-party matching protocol. In this protocol no work from Bob is required, Bob can be offline. The matching protocol is designed to give Alice an indication if Bob is similar to her based on their profiles. We show that the process does so without revealing the private information of Alice and Bob to one another and to the Online Social Network.

Keywords

Online Social Network, profile privacy, profile matching, secret distribution

1. Introduction

Online Social Networks (OSNs) provide an important platform for users to make new friends and share their information. On one hand, OSN users do not fully trust the OSN and users other than their friends, and do not like to disclose their private information to these parties. On the other hand, OSNs may not want the legal responsibility for storing, processing, and distributing the users' private data.

Consider a Facebook user, Alice, who likes a certain Facebook page (for example, a fan page of artist X). On that page, she finds that Bob also likes the same page. Alice thinks he might make a nice friend. To learn more about Bob, she clicks through to his profile. However, Bob's profile is private and Alice does not learn anything more about Bob. Alice now has three options: (1) Give up learning more about Bob. (2) Invite Bob to become friends and hope that Bob will make a good friend. (3) Send Bob a private message and hope that he replies. None of these options both protect Alice's privacy in case that Bob is not a suitable friend (as she has given Bob private information) and allow Alice to learn more about Bob. Either (1) no information is gained, (2) all privacy is lost, or (3) privacy is traded for information. A formal description of the problem can be found in Section 3.

In this paper, we offer a solution that allows Alice to test her similarity with Bob without loss of their privacy. Our solution is based on the following general assumptions:

- A friendship between Alice and Bob is more likely when they have more profile attributes in common (referred to as profile similarity).
- A (randomly chosen) pair of OSN users is unlikely to be online at the same time [1]. So a typical secure two-party computation does not work.
- Users tend to make friends with other users that are in their extended social graph.

We show the intuition behind our solution in Section 4. The building blocks and how we combine them to form our solution can be found in Sections 5 to 7.

Our solution has the following desirable properties:

- It supports offline users by using the OSN as a proxy.
- The profile privacy of Alice is guaranteed by the protocol.
- The profile privacy of Bob relies on the separation of his data between the OSN and Bob's (extended) social graph.

Despite the above achievements, the following challenges remain to be solved.

- Preventing or deterring collusion between the OSN and an OSN user in Bob's (extended) social graph.
- Improving the efficiency for the OSN, so that it can handle a large number of requests simultaneously.

More details of our analysis can be found in Section 8.

2. Related Work

Earlier work attempts to protect user information in OSNs, while preserving the centralized structure. Lucas and Borisov [2] propose to use public key encryption to send messages and profiles between users. Proxy re-encryption is used for group messages. This results in the user holding many keys. Guha et al. [3] propose to shuffle user information across profiles to hide links between information. The OSN is not involved. However, they leave users with no way to verify profile correctness, unless some information is already known. Tootoonchian et al. [4] propose to store profile information at a trusted place. Access is then controlled using relationship certificates. However, this approach removes knowledge of friendship connections. In all these solutions no profile information is available to non-friends for matching scenarios.

3. The Problem Specification

To present a complete overview of the problem we define the terms we are working with, detail the matching scenario, and give the security model for this scenario. With respect to the structure of the OSN, we consider the following terms.

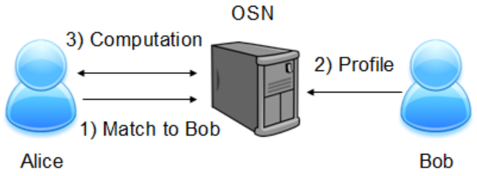

Fig. 1. The matching process without privacy protection

The *OSN* facilitates interaction between users. It stores the user data and acts as a proxy for offline users. The OSN *user* runs a client-side program that is capable, among other things, of performing cryptographic operations. Information pertaining to user X is denoted by a subscript x, for example the profile of X is P_x. A user's *profile* P_x consists of several user attributes. These attributes can be either public or private. For reasons of simplicity, we assume that all attributes are private. However, it is trivial to make attributes public (as normally would be the case for non-sensitive information). The size of the profile is denoted by n_x and is equal to the number of attributes, $n_x = |P_x|$. Within the profile, each attribute can be referenced by $p_{x,i}$, where $1 \leq i \leq n_x$. Each user also maintains a *list of friends* F_x. We assume that friendship connections are symmetrical. This means that if Alice is a friend of Bob, then Bob is also a friend of Alice. The profile, list of friends, and other information are stored on the user's *account*, the user's personal space on the OSN.

3.1. The Matching Scenario

Alice is looking for a new friend. From public information in the OSN she learns that Bob is a potential candidate. To be sure of Bob, Alice runs a matching protocol with the OSN. The OSN represents Bob, this makes the matching process work when Bob is offline. This also prevents Bob from being overloaded with work when there are many users like Alice who want to match with Bob. Fig. 1 shows an overview of the basic matching process without any privacy protection. In the first step, Alice requests the OSN's help in determining a match. In the second step, the OSN retrieves Bob's profile (Bob's data is stored on the OSN). In the third step, Alice and the OSN engage in a two-party matching process. In the simplest version of this, Alice sends her profile to the OSN and the OSN tells Alice if she matches with Bob or not.

We denote information pertaining to Alice with a subscript a, e.g. P_a, and use a subscript b for information pertaining to Bob. Starting with her profile, P_a, Alice defines a threshold t_a specifying when another user is sufficiently similar. If the size of the intersection with Bob's profile, $|P_a \cap P_b|$, is not smaller than this threshold, there is a match.

3.2. Security Model

During the matching process, we distinguish the following five participant groups: the OSN, Alice, Bob, users connected to Alice and Bob (friends), other OSN users. We assume that all groups except other OSN users are honest-but-curious. They will adhere to the protocol specifications, do not provide false information, and do not reveal private information. We further assume that Alice, Bob, and their friends have run the global setup (to be defined in Section 7) and do not collude with the OSN. Everybody is curious about the profiles of Alice and Bob. They will use information gained from the protocol to learn more about the profiles of Alice and Bob (Alice and Bob already know their own profile). For communication between two parties, such as between Alice and the OSN, or between Bob and his friend, we assume there is a secure channel. This means that only the two parties that are involved in the communication process are able to view the messages.

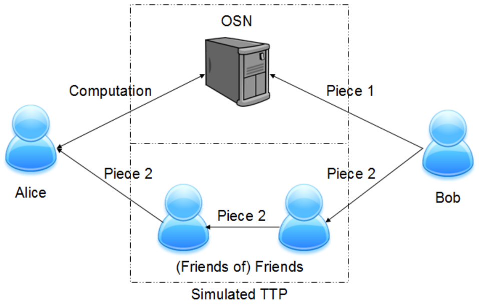

Fig. 2. Simulated trusted third party setting

4. Overview of a Realistic Solution

Ideally, we wish that there is a trusted third party (TTP), fully trusted by the OSN and all OSN users. The matching algorithm then is as follows. In the first step Alice and Bob send their profiles to the TTP. In the second step the TTP computes the size of the profile intersection and sends the result to Alice. In the final step Alice compares the intersection size with the threshold t_a to get a result; *match* when the intersection is larger than or equal to the threshold, or *no match* otherwise. This process can be extended trivially to also inform Bob, the OSN, or both of the result.

Because there is no TTP available in OSNs, we simulate a TTP by using several honest-but-curious parties. An overview of our approach can be found in Fig. 2. The intuition behind our approach is as follows, we separate the profile into two pieces in such a way that neither piece reveals any profile information. Piece 1 will be given to the OSN, piece 2 will be distributed by a chain of friends using a proxy re-encryption scheme.

5. Building Blocks

To create our matching algorithm we use a number of existing techniques. Table 1 explains the used notation. A set of items is denoted with a superscript *.

TABLE 1. Notation

Notation	Explanation	Notation	Explanation
Acc	User Account	$Param$	Security Parameter
$Bool$	Boolean	$RKey$	Re-encryption Key
Cip	Ciphertext	Sec	Secret
$DKey$	Decryption Key	Sig^*	Signature Set
$EKey$	Encryption Key	$SKey$	Signing Key
Msg	Plaintext Message	$VKey$	Verification Key

5.1. Public Key Encryption Scheme

A public key encryption scheme consists of three algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

- Gen : $Param \to (DKey, EKey)$. The key generation algorithm takes a security parameter 1^k as its input and outputs the private key SK and the public key PK.
- Enc : $EKey \times Msg \to Cip$. The encryption algorithm takes the public key PK and a message m as input. The output is the encryption of the message m under this public key PK, that is the ciphertext $c = \mathsf{Enc}_{PK}(m)$.
- Dec : $DKey \times Cip \to Msg$. The decryption algorithm takes the private key SK and a ciphertext c as its input. The output is the message that was encrypted, $m = \mathsf{Dec}_{SK}(c)$, or an error message.

The Paillier's encryption scheme [5] is a homomorphic public key encryption scheme that supports addition on the encrypted messages. The algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ of Paillier are defined as follows:

- Gen : $Param \to (DKey, EKey)$. The key generation algorithm takes a security parameter 1^k as input and generates a tuple (n, p, q, g, λ), where p and q are two primes, $n = pq$, $\lambda = lcm(p-1, q-1)$, and g is a generators of $\mathbb{Z}_{n^2}$. The private key is $SK = \lambda$, and the public key is $PK = (n, g)$.
- Enc : $EKey \times Msg \to Cip$. The encryption algorithm takes a message $m \in \mathbb{Z}_n$ and the public key PK as input. The algorithm outputs the ciphertext $c = g^m r^n \mod n^2$, where $r \in_R \mathbb{Z}_n$.
- Dec : $DKey \times Cip \to Msg$. The decryption algorithm takes a ciphertext c and the private key SK as input, and outputs the message:

  $$m = L(c^\lambda \mod n^2)/L(g^\lambda \mod n^2) \mod n,$$

  where $L(u)$ is defined as $(u-1)/n$.

The scheme is semantically secure under the decisional composite residuosity assumption [5], and it has the following homomorphic property: the result of multiplying two ciphertexts for m and m' is a ciphertext for $m + m'$.

5.2. Proxy Re-Encryption Cryptosystem

When user X and Y each have a key pair for the same public key encryption scheme (which supports re-encryption), then user X can delegate his decryption right to user Y. To facilitate this, the re-encryption process adds two algorithms $(\mathsf{RGen}, \mathsf{REnc})$ to the public key encryption scheme.

- RGen : $DKey \times EKey \to RKey$. The re-encryption key generation algorithm takes the private key SK_x and the public key PK_y as its input. It outputs a re-encryption key $K_{x \to y}$.
- REnc : $RKey \times Cip \to Cip$. The re-encryption algorithm takes a re-encryption key $K_{x \to y}$ and a ciphertext $c = \mathsf{Enc}_{PK_x}(m)$ as its input. The output is the ciphertext $\mathsf{Enc}_{PK_y}(m) = \mathsf{REnc}_{K_{x \to y}}(c)$ which is the message m encrypted under the public key PK_y, or an error message.

In proxy re-encryption [6], the re-encryption keys are stored by a proxy who runs the REnc algorithm on behalf of the users. In the case of a transitive re-encryption scheme, a re-encryption chain can be formed by applying REnc repeatedly.

5.3. Digital Signature Scheme

A public key signature scheme is used to sign messages and consists of three algorithms $(\mathsf{SGen}, \mathsf{Sign}, \mathsf{Ver})$.

- SGen : $Param \to (SKey, VKey)$. The key generation algorithm takes a security parameter 1^k as its input and outputs the signing key GK and the verification key VK.
- Sign : $SKey \times Msg \to Sig$. The encryption algorithm takes the signing key GK and a message m as input. The output is the signature of m, $s = \mathsf{Sign}_{GK}(m)$.
- Ver : $VKey \times Msg \times Sig \to Bool$. The verification algorithm takes the verification key VK, a message m, and a signature s as its input. The output $\mathsf{Ver}_{VK}(m, s)$ is true if s is a valid signature on m under the verification key VK, or false otherwise.

We use these basic cryptographic primitives to construct the protocols for our solution.

6. A New Secret Distribution Scheme for OSN

According to Mislove et al. [7], OSNs typically have a large but weakly connected sub-network. This gives a high probability that there is a path of friends between Alice and Bob. We adopt direct trust propagation [8]: when Alice trusts her friend A, who trusts his friend B, then Alice can trust B. Over multiple friends Alice can eventually trust Bob.

We propose a secret distribution scheme utilizing this type of trust that exists in OSNs. We utilize a cryptographic hash function $H_1 : \{0,1\}^* \to \{0,1\}^k$, that maps a secret to a fixed length output. The secret distribution scheme consists of the following three algorithms $(\mathsf{Setup}, \mathsf{Trans}, \mathsf{Val})$.

Setup : $Param \times Acc^* \times Sec \to ((DKey, EKey), (SKey, VKey), RKey^*, Sig^*, Cip)$. This algorithm is run once by every user X in the OSN and takes a security parameter 1^k, the friends list F_x, and a secret S to distribute as its input. The user first runs the key generation algorithm Gen of a re-encryption cryptosystem to create the key pair (SK_x, PK_x). This key pair forms the basis to distributing the secret. The user then runs the key generation algorithm SGen of a digital signature scheme to create the key pair (GK_x, VK_x). This key pair will be used for validating the secret. The public keys PK_x and VK_x are made public.

Next, for each friend z in the friends list, $z \in F_x$, the user does the following:

- The user runs the RGen algorithm of the re-encryption scheme with inputs the private key SK_x and the friend's public key PK_z and outputs the resulting re-encryption key $K_{x \to z}$. This key will be used to distribute the secret.
- The user signs the friend's verification key VK_z, $\mathsf{Sign}_{GK_x}(VK_z)$. This signature acts as a testimony to the friendship and trust there in. It is used to verify the validity of the distributed secret.
- The user stores the re-encryption key $K_{x \to z}$ and the signed verification key $\mathsf{Sign}_{GK_x}(VK_z)$ in his account.

Note that these actions require the friend to have already created the appropriate key pairs using the Setup algorithm. Finally, the user computes the encryption of the secret S, $\mathsf{Enc}_{PK_x}(S)$, and the signature of the hashed secret S, $\mathsf{Sign}_{GK_x}(H_1(S))$. These are stored in the user's account.

After setup, a user (for example Alice) can ask for the secret of another user (for example Bob). When such a request takes place, the OSN runs the transformation algorithm:

Trans: $Acc \times Acc \times Cip \times RKey^* \times Sig^* \to (Cip, Sig^*)$. The transformation algorithm is run by the OSN and takes as its input two user accounts, i.e. the accounts Bob and Alice, the encrypted secret S encrypted under the public key of Bob, $k_1 = \mathsf{Enc}_{PK_b}(S)$, all re-encryption keys held by the OSN, denoted by $K^* = \{K_{x \to z}, x \in OSN, z \in F_x\}$, and all signatures held by the OSN, denoted by $U^* = \{\mathsf{Sign}_{GK_x}(H_1(S)), x \in OSN\} \cup \{\mathsf{Sign}_{GK_x}(VK_z), x \in OSN, z \in F_x\}$. The OSN takes the following steps:

- Find a path from Bob to Alice. The OSN knows how all users are connected, as it has key material associated with each link. Finding a path is then done through traditional path discovery methods in graph theory [9]. Note that there are likely to be many paths and the resulting path is not necessarily the shortest. The length of the path (including Alice and Bob) is denoted by l.
- Use the REnc algorithm with the first key in the path and k_1 as input. The new output k_i is iteratively used with the i'th key in the path as input for the re-encryption algorithm REnc (creating a re-encryption chain), until the end is reached, $1 < i < l$. The last ciphertext k_l is the original secret encrypted under the public key of Alice, $k_l = \mathsf{Enc}_{PK_a}(S)$.
- Create a set of signatures V^*. This set contains the signature of the secret $\mathsf{Sign}_{GK_b}(H_1(S))$, as output by the Distr algorithm, and the signatures of the verification keys of the users along the path. If the first step in the path is from Bob to his friend Zara the corresponding signature would be $\mathsf{Sign}_{GK_z}(VK_b)$, this signature is created by Zara in the Setup algorithm.
- Output the encrypted secret k_l and the signature set V^*.

When Alice receives an encrypted secret and a signature set from the OSN, she proceeds to validate the encrypted secret using the validation algorithm:

Val: $DKey \times Cip \times VKey^* \times Sig^* \to Sec$. The key validation algorithm extracts the distributed secret and validates it. This algorithm is run by Alice and takes as input the private key SK_a, an encrypted secret $k_l = \mathsf{Enc}_{PK_a}(S)$, all public verification keys, denoted by $VK^* = \{VK_x, x \in OSN\}$, and the signature set V^* as created by the OSN in the Trans algorithm. It outputs the secret S or an error. Alice first decrypts k_l using the decryption algorithm and her private key SK_a, $S = \mathsf{Dec}_{SK_a}(k_l)$. Alice then checks all the signatures in the signature set V^* (the signed hash $\mathsf{Sign}_{GK_b}(H_1(S))$ and the signed verification keys, for example $\mathsf{Sign}_{GK_z}(VK_b)$) using the signature verification algorithm Ver. This creates a chain of verified verification keys (and thus a chain of trust), that starts with the verification key of Alice, VK_a. Only when Alice can decrypt the secret S and all verifications pass will this algorithm output the secret S.

7. The Proposed Matching Solution

The matching process consists of a global setup and a matching protocol. The global setup is run only once, during which each user pre-computes required values. The two-party matching protocol allows a user (for example Alice) to determine the existence of a match with another user (for example Bob) using the OSN as a proxy. This can be done multiple times.

7.1. The Global Setup

The global setup of the matching process is run by each user, for example Bob. Bob first hashes his profile, $HP_b = \{H_2(p_{b,i}), 1 \leq i \leq n_b\}$, using the cryptographic hash function $H_2 : \{0,1\}^* \to \{0,1\}^k$ that maps profile attributes to a fixed length output. Bob then runs a polynomial creation algorithm with this hashed profile HP_b as input. This creates as output the polynomial $Q_b(x)$ that has the attributes from the hashed profile HP_b as its roots. Bob then generates a random polynomial $R_b(x)$ and a polynomial $S_b(x)$ such that $Q_b(x) = R_b(x) + S_b(x)$. Neither $R_b(x)$ nor $S_b(x)$ alone reveals any information about the profile polynomial $Q_b(x)$, so they can each be revealed to a different semi-trusted party.

Bob then runs the setup algorithm Setup from the key distribution scheme with as inputs the security parameter 1^k, the friends list F_b, and the polynomial $S_b(x)$ that is to be distributed.

Bob stores the private keys SK_b and GK_b outside the OSN. Finally, as output Bob stores in his account the public keys PK_b, and VK_b, all re-encryption keys, $K_{b \to z}$ and all signatures of his friends' verification keys, $\mathsf{Sign}_{GK_b}(VK_z)$, where $z \in F_b$, the encrypted polynomial, $\mathsf{Enc}_{PK_b}(S_b(x))$, the signature of the hashed polynomial, $\mathsf{Sign}_{GK_b}(H_1(S_b(x)))$, and the polynomial $R_b(x)$.

7.2. The Matching Protocol

The proposed matching protocol is an adaptation of the FNP set intersection cardinality protocol, proposed by Freedman, Nissim and Pinkas [10]. Instead of one party holding the set polynomial, now both parties hold a part of the polynomial. This replacement transforms the FNP protocol from a two-party interactive protocol to a non-interactive protocol utilizing a third party. The proposed protocol uses the new key distribution scheme, proposed in Section 6, to securely transfers the polynomial $S_b(x)$ from Bob to Alice.

The matching protocol consists of two phases. In the first phase the polynomial $S_b(x)$ is transferred from Bob to Alice. In the second phase the actual computation takes place.

7.2.1. Phase 1: Distribution of the polynomial $S_b(x)$ from Bob to Alice. The first phase of the protocol is as follows.

1) Alice tells the OSN that she wants to match with Bob.
2) The OSN uses the transformation algorithm Trans from the secret distribution scheme with inputs the accounts of Bob and Alice, Bob's encrypted polynomial $\mathsf{Enc}_{PK_b}(S_b(x))$, the set of re-encryption keys K^*, and the set of signatures U^*. Both sets are as detailed in the transformation algorithm Trans. The outputs of this algorithm are the re-encrypted polynomial, $\mathsf{Enc}_{PK_a}(S_b(x))$, and set of signatures V^*. V^* contains the signatures on the hashed polynomial, $\mathsf{Sign}_{GK_b}(H_1(S_b(x)))$, and verification keys from the path between Alice and Bob, for example $\mathsf{Sign}_{GK_z}(VK_b)$. The output is sent back to Alice.
3) Alice validates Bob's polynomial $S_b(x)$ using the secret validation algorithm Val. This algorithm takes as input Alice's private key SK_a, the encrypted polynomial $\mathsf{Enc}_{PK_a}(S_b(x))$, all verification keys VK^*, and the signature set V^* received from the OSN. As output Alice receives Bob's polynomial $S_b(x)$.

7.2.2. Phase 2: Two-party similarity computation. An overview of phase 2 of the protocol is shown in Fig. 3. In more details, the proposed protocol is described as follows.

1) The OSN creates a new ephemeral key pair for the Paillier cryptosystem. This key pair consists of the private key DK_S and the public key EK_S. The message space is $\mathbb{Z}_S$. Next the OSN uses the public key to encrypt the coefficients c_j, where $0 \le j \le n_b$, of the polynomial $R_b(x)$. These encryptions will be used to calculate a result for $R_b(x)$ as in the FNP protocol. For simplicity we assume that $|R_b(x)| = n_b$. The public key EK_S, degree of the polynomial n_b, and the encryptions q_j, where $0 \le j \le n_b$, are sent to Alice.
2) Alice also creates a new ephemeral key pair for the Paillier cryptosystem. This key pair consists of the private key DK_a and the public key EK_a. The message space is $\mathbb{Z}_a$. For each attribute in her profile, $p_{a,i}, 1 \le i \le n_a$, she hashes the attribute $H_2(p_{a,i})$ and creates two random values r_i and r'_i. She then computes $Q_b(H_2(p_{a,i})) = R_b(H_2(p_{a,i})) + S_b(H_2(p_{a,i}))$. This is done by obliviously computing $R_b(H_2(p_{a,i}))$, computing $S_b(H_2(p_{a,i}))$ in the clear, and adding the two together. The result is then randomized using r_i and r'_i as follows:

$$t_i = \mathsf{Enc}_{EK_S}(Q_b(H_2(p_{a,i})) \cdot r_i + r'_i) =$$

$$((\prod_{j=1}^{n_b} q_j^{H_2(p_{a,i})^j}) \cdot \mathsf{Enc}_{EK_S}(S_b(H_2(p_{a,i}))))^{r_i} \cdot \mathsf{Enc}_{EK_S}(r'_i)$$

Alice also encrypts the negation of the second random value r'_i under her own public key, $u_i = \mathsf{Enc}_{EK_a}(-r'_i)$. Alice sends to the server her public key EK_a, the size of her profile n_a, the encrypted and randomized polynomial results t_i, and the encrypted random values u_i, where $1 \le i \le n_a$.

Alice $(P_a, n_a, t_a, S_b(x))$ OSN $(R_b(x), n_b)$

$(DK_S, EK_S) = \mathsf{Gen}(1^k)$
$q_j = \mathsf{Enc}_{EK_S}(c_j), 0 \le j \le n_b$

$\xleftarrow{EK_S, n_b, q_j, 0 \le j \le n_b}$

$(DK_a, EK_a) = \mathsf{Gen}(1^k)$
$\forall p_{a,i}, 1 \le i \le n_a:$
$r_i, r'_i \in_r \mathbb{Z}_S$
$t_i = \mathsf{Enc}_{EK_S}(Q_b(H_2(p_{a,i})) \cdot r_i + r'_i)$
$u_i = \mathsf{Enc}_{EK_a}(-r'_i)$

$\xrightarrow{EK_a, n_a, t_i, u_i, 1 \le i \le n_a}$

$\forall (t_i, u_i), 1 \le i \le n_a:$
$r''_i \in_r \mathbb{Z}_a$
$t'_i = \mathsf{Enc}_{EK_a}(\mathsf{Dec}_{DK_S}(t_i))$
$e_i = (t'_i \cdot u_i)^{r''_i}$

$\xleftarrow{e_i, 1 \le i \le n_a}$

$d_i = \mathsf{Dec}_{DK_a}(e_i), 1 \le i \le n_a$
$s = \#(d_i = 0), 1 \le i \le n_a$
if $t_a \le s$ **then**
match **else** *no match*

Fig. 3. Phase 2 of the matching algorithm

3) For each polynomial result and random value pair the OSN creates a random value r''_i. The OSN then re-encrypts and randomizes the result as follows:

$$e_i = (\mathsf{Enc}_{EK_a}(\mathsf{Dec}_{DK_S}(t_i)) \cdot u_i)^{r''_i}$$

The final encrypted values, $e_i, 1 \le i \le n_a$, are sent back to Alice.
4) Alice decrypts the results using her private DK_a and checks for occurrences of '0'. A '0' indicates an attribute that is in the set intersection as in the FNP protocol. If at least t_a elements decrypt to '0', then there is a match.

8. Security of the Solution

In the ideal situation Alice only learns the size of the profile intersection and as a consequence whether there is a match or not. However, in our scheme the degree of the polynomials reveals Bob's profile size and the number of pairs the OSN receives from Alice reveals her profile size. Revealing the profile sizes to the OSN only reveals the potential richness of the profile (a larger profile is likely more detailed). Revealing Bob's profile size to Alice results in her knowing the number of attributes in Bob's profile that she does not have in common with him, $n_b - |P_a \cap P_b|$. The profile sizes reveal no information about the content of the profiles.

8.1. Security of Alice's Profile

The only information the OSN receives from Alice are blinded results to a polynomial that the OSN does not know and an encrypted value to remove the blinding. Clearly the blinded result does not give the OSN any information about the input to the polynomial. In order to remove the blinding the OSN must first encrypt using Alice's public key EK_a. Because the key pair is fresh and the Paillier cryptosystem is semantically secure [5], the encryptions (both to unblind and the polynomial result after unblinding) do not reveal any information to the OSN. Bob's only influence on the protocol is through his profile. He does not receive any messages during the execution and can thus not learn anything about Alice's profile. The same goes for friends and other OSN users, except they don't have any influence.

8.2. Security of Bob's Profile

After the protocol Alice has access to the following items, the polynomial $S_b(x)$, an encrypted version of the polynomial $R_b(x)$, and the result values $Q_b(H_2(p_{a,i})) \cdot r_i^*$. As in the FNP protocol, the result values reveal no information about the polynomial they are based on $Q_b(x)$. For each hashed attribute, $x_i = H_2(p_{a,i}), p_{a,i} \in P_a$, that Alice uses as input the result of the polynomial $Q_b(x_i)$ is computed. The result is either a 0 if x_i is a root of the polynomial (x_i is in Bob's profile), or non 0 otherwise. By multiplying this with a random number r (r can not be 0) the result remains 0 if x_i is a root of the polynomial, or a random number otherwise. When this result is returned to Alice she either receives 0, and learns that her input attribute is also an attribute in Bob's profile, or a random value, and learns that her input was not an attribute in Bob's profile (and nothing else). Thus the only way to learn the polynomial $Q_b(x)$, is by combining the polynomials $R_b(x)$ and $S_b(x)$. Alice has the polynomial $S_b(x)$. However, she only has access to the encrypted version of the polynomial $R_b(x)$. The key used for the encryption is a fresh one (so it is not compromised) and the Paillier cryptosystem is semantically secure, so the encryption leaks no information about it's contents. Alice cannot retrieve the polynomial $R_b(x)$.

The OSN has access to the polynomial $R_b(x)$. This polynomial reveals no information to the OSN about Bob's profile as long as the OSN does not have Bob's other polynomial $S_b(x)$. The OSN has access to the encrypted version of the polynomial $S_b(x)$ and a large number of re-encryption keys. A re-encryption scheme hides the message from the proxy (the OSN), thus the OSN will not learn anything. Because no user (that is connected through a chain of friends to Bob) colludes with the OSN, the OSN does not have access to a private key and a matching public key encryption of $S_b(x)$. Bob's polynomial $S_b(x)$ and thus Bob's profile polynomial $Q_b(x)$ are secure.

Friends and other OSN users have no input to the protocol and do not receive any messages during the execution. Thus they can not learn any information about Bob's profile. The OSN is responsible for storing both polynomials (one in plain and one encrypted). Should a user, say Eve (who is an indirect friend of Bob), collude with the OSN, than together they can recover Bob's profile polynomial. Preventing or deterring such collusion is left to future work.

9. Conclusion

We offer a construction with which users can test their similarity without a need for lowering privacy protection. This alternative is accomplished by combining secure multi-party computation with the social graph, and its implied trust. As a result we have a matching process that is secure in the honest-but-curious model, and works in a realistic OSN setting where a pair of users is unlikely to be online at the same time. It remains as future work to find a solution in a model where some friends are malicious.

References

[1] F. Benevenuto, T. Rodrigues, M. Cha, and V. A. F. Almeida, "Characterizing user behavior in online social networks." in *Internet Measurement Conference*, 2009, pp. 49–62.

[2] M. M. Lucas and N. Borisov, "Flybynight: mitigating the privacy risks of social networking," in *7th ACM workshop on Privacy in the electronic society (WPES)*, 2008, pp. 1–8.

[3] S. Guha, K. Tang, and P. Francis, "Noyb: privacy in online social networks," in *First workshop on Online Social Networks (WOSP)*, 2008, pp. 49–54.

[4] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman, "Lockr: better privacy for social networks," in *5th international conference on Emerging networking experiments and technologies*, 2009, pp. 169–180.

[5] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT '99*, 1999, pp. 223–238.

[6] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *EUROCRYPT'98*, ser. Lecture Notes in Computer Science, K. Nyberg, Ed., 1998, vol. 1403, pp. 127–144.

[7] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *7th ACM SIGCOMM conference on Internet measurement*, 2007, pp. 29–42.

[8] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins, "Propagation of trust and distrust," in *13th international conference on World Wide Web*, 2004, pp. 403–412.

[9] D. B. West, *Introduction to Graph Theory*, 1996.

[10] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *EUROCRYPT '04*, 2004, pp. 1–19.