Fuzzy Private Matching

Łukasz Chmielewski
Security of Systems (SoS) group
Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, the Netherlands
lukaszc@cs.ru.nl

Jaap-Henk Hoepman
TNO ICT
P.O. Box 1416, 9701 BK Groningen, The Netherlands
jaap-henk.hoepman@tno.nl, and
Security of Systems (SoS) group
Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, the Netherlands
jhh@cs.ru.nl

Abstract

In the private matching problem, a client and a server each hold a set of n elements as their input. The client wants to privately compute the intersection of these two sets: the client learns the elements it has in common with server (and nothing more), while the server obtains no information at all. In certain applications it would be useful to have a private matching protocol that reports a match even if two elements are only similar instead of equal. Such a private matching protocol is called *fuzzy*, and is useful, for instance, when elements may be inaccurate or corrupted by errors.

We consider the fuzzy private matching problem, in a semi-honest environment. Elements are similar if they match on t out of T attributes. First we show that the original solution proposed by Freedman *et al.* [FNP04] is not private: the client can "steal" elements even if the sets have *no* similar elements in common. Following that we present 2 fuzzy private matching protocols. The first, simple, protocol has message complexity $O(n\binom{T}{t})$. The second, improved, protocol has message complexity $O(nT)$, but here the client incurs a $O(n^2\binom{T}{t})$ time complexity penalty.

1 Introduction

In the private matching problem [FNP04], a client and a server each hold a set of elements as their input. The size of the set n and the type of elements is publicly known. The client wants to privately compute the intersection of these two sets: the client learns the elements it has in common with server (and nothing more), while the server obtains no information at all. This problem is related to the set intersection problem, except that one of the parties is not allowed to learn even the elements within the intersection (see below). Efficient protocols solving this problem are known.

In certain applications, the elements (think of them as words consisting of letters, or tuples of attributes) may not always be accurate or completely known. For example, due to errors, omissions, or inconsistent spelling, entries in a database may not be identical. In these cases, it would be useful to have a private matching algorithm that reports a match even if two entries are only similar. Such a private matching is called *fuzzy*, and was introduced by Freedman *et* *al.* [FNP04]. Elements are called similar in this context if they match on t out of T attributes (or t out of T letters).

Fuzzy private matching (FPM) protocols could also be used to implement a more secure and private form of biometric pattern matching. Instead of sending over the complete template corresponding to say a scanned finger, a fuzzy private matching protocol could be used to determine the similarity of the scanned finger with the templates stored in the database, without revealing any information about this template in case no match is found.

Freedman *et al.* [FNP04] introduce the fuzzy private matching problem and present a protocol for 2-out-of-3 fuzzy private matching. We show that, unfortunately, this protocol is incorrect (see Section 3): the client can "steal" elements even if the sets have *no* similar elements in common. Building and improving on their ideas, we present two protocols for t-out-of-T fuzzy private matching (henceforth simply called fuzzy private matching). The first, simple, protocol has message complexity $O(n\binom{T}{t})$. The second, improved, protocol is based on linear secret sharing and has message complexity $O(nT)$. Here the client incurs a $O(n^2\binom{T}{t})$ time complexity penalty, however.

Indyk and Woodruff [IW06] present another approach for solving fuzzy private matching, using generic techniques like secure 2-party computation and oblivious transfer. To compare their results to ours, we use their notation to express the bit complexity of the protocols: they define $f = \tilde{O}(g)$ if $f(n,k) = O\left(g(n,k)\log^{O(1)}(n)\text{poly}(k)\right)$, where k is the security parameter.

Solutions based on secure function evaluation (using generic secure 2-party computation) work in bit complexity $\tilde{O}(n^2T)$, while the solution of Indyk and Woodruff [IW06] works in $\tilde{O}(nT^2+n^2)$. The Freedman *et al.* [FNP04] protocol (though incorrect), as well as our first corrected version work in bit complexity $\tilde{O}(n\binom{T}{t})$. In comparison, our most efficient protocol works in bit complexity $\tilde{O}(nT)$ (however with the aforementioned increased time complexity of the client). Moreover, our protocols (including the one based on protocols from [FNP04]) do not use generic secure 2-party computation constructions or oblivious transfer protocols. Because of that they are more efficient than would appear from the $\tilde{O}$ notation (see above).

Related work can be traced back to private equality testing [BST01, FNW96, FNP04, NP99] in the 2-party case, where each party has a single element and wants to know if they are equal (without publishing these elements). Private set intersection [FNP04, NP99, KS05] (possibly among more than two parties) is also related. In this problem the output of *all* the participants should be the intersection of all the input sets, but nothing more: a participant should gain no knowledge about elements from other participant's sets that are not in the intersection.

Similarly related are so called secret handshaking protocols [BDS$^+$03, CJT04]. They consider membership of a secret group, and allow members of such groups to reliably identify fellow group members without giving away their group membership to non-members and eavesdroppers. We note that the (subtle) difference between secret handshaking and set-intersection protocols lies in the fact that a set-intersection protocol needs to be secure for arbitrary element domains (small ones in particular), whereas group membership for handshaking protocols can be encoded using specially constructed secret values taken from a large domain.

Privacy preservation issues have also been considered for approximation of a function f among vectors owned by several parties. The function f may be Euclidean distance ([DA00], [FIM$^+$01], [IW06]), set difference ([FNP04]), Hamming distance ([DA00], [IW06]), and scalar product (reviewed in [GLLM04]).

Our paper is structured as follows. We formally define the fuzzy private matching problem in Section 2, and introduce our system model, some additional notation, and primitives there as well. Then in section 3 we present the solution from [FNP04] for 2-out-of-3 fuzzy private matching and show where it breaks. Section 4 contains our first protocol for t-out-of-T fuzzy private matching that uses techniques similar to the ones used in [FNP04]. We then present our second protocol based on linear secret sharing in section 5. All our protocols assume a semi-honest environment (see Section 2.4).

2 Preliminaries

In this section, we introduce the fuzzy matching problem as well as the mathematical and cryptographic tools that we use to construct our protocols.

2.1 Fuzzy Private Matching Problem Definition

Let a client and a server each own a set of words. A fuzzy private matching scheme is a 2-party protocol between a client and a server, that allows the client to compute the fuzzy set intersection of these sets (without leaking any information to the server).

To be precise, let all the words $X = x^1 \dots x^T$ in these sets consist of T letters x^i from a domain D. We define an auxiliary relation $X \approx_t Y$ among these words as follows: we say that two words $X = x^1 \dots x^T$ and $Y = y^1 \dots y^T$ match on t letters if and only if

$$t \leq |\{k : x^k = y^k \cap (1 \leq k \leq T)\}| \ .$$

The input and the output of the protocol are defined as follows.

input:

- For the client: set $X = \{X_1, \dots X_{n_C}\}$ of n_C words of length T.
- For the server: set $Y = \{Y_1, \dots Y_{n_S}\}$ of n_S words of length T.
- For both client and server: n_C, n_S, T and t.

output:

- Output of the client is a set: $\{Y_i \in Y | \exists X_i \in X : X_i \approx_t Y_j\}$. This set consist of all the elements from Y that match with any element from the set X.
- Output of the server is empty (the server does not learn anything).

Usually we assume that $n_C = n_S = n$. In any case, the sizes of the sets are fixed and known to the other party a priori (so the protocol does not have to prevent the other party to learn the size of the set).

2.2 Additively Homomorphic Cryptosystem

We use a semantically-secure, additively homomorphic public-key cryptosystem, e.g., Paillier's system [Pai99]. Let $\{\cdot\}_{pk}$ denote the encryption function with public key pk. The homomorphic cryptosystem supports the following two operations, which can be performed without the knowledge of the private key.

1. Given the encryptions $\{a\}_{pk}$ and $\{b\}_{pk}$, of a and b, one can efficiently compute the encryption of $a + b$, denoted $\{a + b\}_{pk} := \{a\}_{pk} +_h \{b\}_{pk}$
2. Given a constant c and the encryption $\{a\}_{pk}$, of a, one can efficiently compute the encryption of $c \cdot a$, denoted $\{c \cdot a\}_{pk} := c \cdot_h \{a\}_{pk}$

These properties hold for suitable operations $+_h$ and $\cdot_h$ defined over the range of the encryption function. In Paillier's system, operation $+_h$ is a multiplication and $\cdot_h$ is an exponentiation.

2.2.1 Operations on encrypted polynomials

We represent any polynomial p of degree n (on some ring) as an ordered list of its coefficients: $[\alpha_0, \alpha_1, \dots, \alpha_n]$. We denote the encryption of a polynomial p as $\{p\}_{pk}$ and define it to be the list of encryptions of its coefficients: $[\{\alpha_0\}_{pk}, \{\alpha_1\}_{pk}, \dots, \{\alpha_n\}_{pk}]$.

Many possible operation can be performed on such encrypted polynomials (assuming that the encryption has an additively homomorphic property), like: addition of two encrypted polynomials or multiplication of an encrypted and a plain polynomial. However in this paper we use the following property: given an encryption of a polynomial $\{p\}_{pk}$ and some x one can efficiently compute a value $\{p(x)\}_{pk}$. This follows from the properties of the homomorphic encryption scheme:

$$\{p(x)\}_{pk} = \left\{ \sum_{i=0}^{n} \alpha_i \cdot x^i \right\}_{pk} = {\sum_{i=0}^{n}}_h \{\alpha_i \cdot x^i\}_{pk} = {\sum_{i=0}^{n}}_h \{\alpha_i\}_{pk} \cdot_h x^i$$

2.3 Linear Secret Sharing

Secret sharing refers to any method for distributing a secret among a group of n participants, each of which is allocated a share of the secret. The secret can only be reconstructed when at least t shares are combined together. Combining less than t individual shares are of no use and should give no information whatsoever about the secret. We denote a secret share as $\overline{s}_i$ (for $i \in \{1 \dots n\}$) and the corresponding secret as $\overline{s}$.

A Linear Secret Sharing (LSS) scheme is a secret sharing scheme with additional properties. In this paper we use the following property: given t shares $\overline{s}_i$ (of secret $\overline{s}$), and t shares $\overline{r}_i$ (of secret $\overline{r}$) on the same indecies, using $\overline{s}_i + \overline{r}_i$ one can reconstruct the sum of the secrets $\overline{s} + \overline{r}$. Such a LSS scheme is Shamir's original secret sharing scheme [Sha79].

2.4 Adversary Models

In this section we describe the adversary model that we use. We prove correctness of our protocols only against a semi-honest adversary (we do not consider a malicious one). Here we provide the intuition and the informal notion of this model, the reader is referred to [Gol02] for full definitions. To simplify matters we only consider the case of only two participants (the client and the server).

In the model with a semi-honest adversary, both parties are assumed to act accordingly to their prescribed actions in the protocol (but they are allowed to use all information that they collect in an unexpected way to obtain extra information). The security definition is straightforward in our particular case, as only one party (client) learns the output. Following [FNP04] we divide the requirements into:

- The client's security – **indistinguishably**:
  Given that the server gets no output from the protocol, the definition of the client's privacy requires simply that the server cannot distinguish between cases in which the client has different inputs.

- The server's security – **comparison to the ideal model**:
  The definition ensures that the client does not get more or different information than the output of the function. This is formalized by considering an ideal implementation where a trusted third party TTP gets the inputs of the two parties and outputs the defined function. We require that in the real implementation of the protocol (one without TTP) the client does not learn different information than in the ideal implementation.

Due to space constraints our proofs are informal, presenting only the main arguments for correctness and security.

3 The Original FPM Protocol

In this section we present the original fuzzy private matching protocol from Freedman *et al.* [FNP04] (pages: 16–17). We show (following the original paper) the version for $T = 3$ and $t = 2$. Then we present an example of an input data where this protocol fails. The protocol is presented in Figure 1.

Figure 1: FPM protocol – version from [FNP04]

Domain remark: The domain R of the plaintext of the homomorphic cryptosystem is defined as follows: R should be bigger than D^T and the following property should hold: "a random element from R is with negligible probability in D^T". This property can be satisfied by representing an element $a \in D^T$ by $r_a = 0^k||a$ in R.

1. The client chooses a private key sk, a public key pk and *parameters* for homomorphic encryption scheme and sends pk and *parameters* to the server.
2. The client:
   (a) chooses, for every i (such that $1 \leq i \leq n_C$), a random value $r_i \in R$
   (b) creates 3 polynomials: P_1, P_2, P_3 (where polynomial P_j is used to encode all letters on the jth position) defined by the set of equations $r_i = P_1(x_i^1) = P_2(x_i^2) = P_3(x_i^3)$, for $1 \leq i \leq n_C$
   (c) uses interpolation to calculate coefficients of the polynomials (P_1, P_2, P_3) and sends their encryptions to the server.
   Remark: These polynomials have degree $n_C - 1$ (in [FNP04] it is written that they have degree n_C).
3. For each Y_j (such that $1 \leq i \leq n_S$), the server responds to the client:
   $\{r \cdot (P_1(y_i^1) - P_2(y_i^2)) + Y_i\}_{pk}$,
   $\{r' \cdot (P_2(y_i^2) - P_3(y_i^3)) + Y_i\}_{pk}$,
   $\{r'' \cdot (P_1(y_i^1) - P_3(y_i^3)) + Y_i\}_{pk}$,
   where r, r', r'' are always fresh random values in R. This uses the properties of the homomorphic encryption scheme and the use of encrypted polynomials explained in section 2.2.1.
4. If the client receives encryption of encoding of Y_i, which is similar to any word from his set X then he adds it to the output set.

3.1 The idea behind, and the problem of protocol 1:

Intuitively the protocol works because if $X_i \approx_2 Y_j$ then, say, $x_i^2 = y_j^2$ and $x_i^3 = y_j^3$. Hence $P_2(x_i^2) = P_2(y_j^2) = r_i$ and $P_3(x_i^3) = P_3(y_j^3) = r_i$ so $P_2(y_i^2) - P_3(y_j^3) = 0$. Then the result $\{r' \cdot (P_2(y_j^2) - P_3(y_j^3)) + Y_j\}_{pk}$ sent back by the server simplifies to $\{Y_j\}_{pk}$ (the random value r' is canceled by the encryption of 0) which the client can decrypt. If X_i and Y_j do not match, the random values r, r' and r'' do not get canceled and effectively blind the value of Y_j in the encryption, hiding it to the client.

There is a problem with this approach however. Consider the following proper input data.

Client's INPUT:	Server's INPUT:
$\{[1,2,3]\ ,\ [1,4,5]\}$	$\{[5,4,3]\}$

Then in step 2c of the protocol 1, the polynomials are defined (by the client) in the following way:

$\mathbf{P}_1$:	$\mathbf{P}_2$:	$\mathbf{P}_3$:
$P_1(1) = r_1 \cap P_1(1) = r_2$	$P_2(2) = r_1 \cap P_2(4) = r_2$	$P_3(3) = r_1 \cap P_3(5) = r_2$

But now we see that, unless $r_1 = r_2$ (which is unlikely when they are both chosen at random), P_1 remains undefined! Freedman *et al.* do not consider this possibility. However, if we try to remedy this problem by setting $r_1 = r_2$ we obtain another one. Among other things, the server computes $\{r' \cdot (P_2(y_i^2) - P_3(y_i^3)) + Y_i\}_{pk}$, which, in this particular case equals $\{r' \cdot (P_2(4) - P_3(3)) + [5,4,3]\}_{pk}$. This equals $\{r' \cdot (r_2 - r_1) + [5,4,3]\}_{pk}$, which by equality of r_1 and r_2 reduces to $\{[5,4,3]\}_{pk}$. In other words, the client learns $[5,4,3]$ even if this value does not match any of the elements held by the client. This violates the requirements of the fuzzy private matching problem.

4 Polynomial Based Protocol for FPM problem

In this section we present our protocol solving private fuzzy matching problem inspired by the protocol from [FNP04] (presented in Section 3). Our protocol works for any T and t. The protocol is presented in Figure 2. In the protocol we use the following definition of σ. Let σ be a combination of t different indecies $\sigma_1, \sigma_2, \ldots, \sigma_t$ from the range $\{1, \ldots, T\}$ (there are $\binom{T}{t}$ of those). For a word $X \in D^T$, define $\sigma(X) = x^{\sigma_1} || \cdots || x^{\sigma_t}$ (i.e., the concatenation of the letters in X found at the indecies in the combination).

Figure 2: Corrected Protocol solving private fuzzy matching problem.

Domain remark: like in the protocol 1.

1. The client chooses a private key sk, a public key pk and *parameters* for homomorphic encryption scheme and sends pk and the *parameters* to the server.
2. For every combination σ of t out of T indecies the client:
   (a) constructs polynomial:
   $P_\sigma(x) = (x - \sigma(X_1)) \cdot (x - \sigma(X_2)) \cdots (x - \sigma(X_{n_C}))$ of degree n_C
   (b) sends $\{P_\sigma\}_{pk}$ (the encrypted polynomial, see section 2.2.1) to the server.
3. For every $Y_i \in Y$, $1 \leq i \leq n_S$, the server performs:
   (a) for every received polynomial $\{P_\sigma\}_{pk}$ (corresponding to the combination σ):
      i. evaluate polynomial $\{P_\sigma\}_{pk}$ at the point $\sigma(Y_i)$ to compute $\{w_i^\sigma\}_{pk} = \{r * P_\sigma(\sigma(Y_i)) + Y_i\}_{pk}$, where $r \in R$ is always a fresh random value.
      ii. sends $\{w_i^\sigma\}_{pk}$ to the client.
4. The client decrypts all received messages and if a received message $w_i^\sigma \in D^T$ and it matches with any word from X then he adds w_i^σ to the output set.

4.1 Correctness of protocol 2

In the protocol the client produces $\binom{T}{t}$ polynomials P_σ of degree n_C. Every polynomial represents one of the combinations σ of t letters from T letters. It is easy to see that if $X \approx_t Y$ then $\sigma(X) = \sigma(Y)$ for some such combination σ. The roots of each polynomial are concatenated letters (of every word in the client set) corresponding to each combination. Hence, if in the set Y there is an element Y_j that matches with any $X_i \in X$, then in step 3(a)i of the protocol the value of the evaluated polynomial at a “matching point” σ is 0 and then the encryption of Y_j is sent to the client. Afterwards the client can recognize this value. Otherwise (if Y_j does not match with any element form X) then all of the values sent to the client contain a random blinding element r (and therefore their decryptions are in X with negligible probability).

4.2 Security of protocol 2

Client input data is secure because all of the data received by the server are encrypted (using a semantically secure cryptosystem). Hence the server cannot distinguish between different client input.

Privacy of the server is protected because the client learns about those elements from Y that are also in X (if element $y_i \in Y$ does not belong to X then a random value is sent by the server, see the previous section).

4.3 Complexity

Messages being sent in this protocol are encryptions of plaintext from a domain that contains D^T, enlarged by k bits (where k is the security parameter). In step 2 the client sends $\binom{T}{t}$ polynomials of degree n_C. Then in step 3 the server responds with n_S values for every polynomial. Hence in total $O((n_S + n_C) \cdot \binom{T}{t})$ messages are send. Time complexities of the participants also contain factor $\binom{T}{t}$ (which follows from the bit complexity).

4.4 Optimization for a domain of messages

A domain of messages for large D and T can significantly slow down the bit performance. However there is a way to make this domain smaller by slightly modifying the protocol. For every Y_i the server should prepare an unique secret key sk_i and public key pk_i. Then for every Y_i he sends $E_{pk_i}(0^k||Y_i)$ to the client. After that, the protocol should be run unchanged, except that in step 3(a)i the server calculates and sends $\{w_i^\sigma\}_{pk} = \{r * P_\sigma(\sigma(Y_i)) + (0^k||sk_i)\}_{pk}$ instead. Later in step 3(a)i the client can distinguish a valid secret key from the random value (by the prefix 0^k) and check to which encryption $E_{pk_i}(0^k||Y_i)$ it fits. After he finds such an encryption he can add it to his output set.

In this modified protocol $O((n_S + n_C) \cdot \binom{T}{t})$ messages from a domain of size $O(k)$ are sent and $O(n_s)$ messages from a domain of size $O(log(|D|^T) + k)$.

4.5 Remarks

All of the optimizations described in [FNP04] used for private matching problem can be easily used in this protocol.

It is also easy to modify our protocol to be resistant to a malicious adversary (using the protocol resistant to a malicious adversary from [FNP04]). Every polynomial should be protected from a malicious adversary separately (it is a similar situation to $\binom{T}{t}$ instances of private matching problem against a malicious adversary).

5 Secret Sharing Based Protocol for FPM problem

In this section we present two of our protocols solving FPM problem. Both of them use the linear secret sharing technique (described in Section 2.3) and work in the model with semi-honest adversary. First we describe simple (but slow) protocol and later faster, improved one.

5.1 A simple version of the protocol

The simple protocol is presented in Figure 3. In this protocol the client first sends encryptions of all of his words (every letter is encrypted separately) to the server. Then the participants, for every couple of words from X and Y, run subroutine `find-matching`(i,j). The aim of a single call of this procedure is to provide Y_j to the client if and only if $X_i \approx_t Y_j$. This is achieved by using t–out–of–T secret sharing scheme. The client receives a correct share from the server if corresponding letters x_i^w and y_j^w are equal (otherwise he receives random value). Hence he can recover Y_i if he receives at least t correct shares (and this happens if and only if at least t letters from X_i are equal to Y_j).

Figure 3: Simple protocol solving private fuzzy matching problem

1. The client generates sk, pk and $parameters$ for a semantically secure, additively homomorphic cryptosystem and sends pk and $parameters$ to the server.
2. For each $X_i \in X$
   (a) The client encrypts each letter x_i^w of X_i and sends $\{x_i^w\}_{pk}$ to the server.
   (b) For each $Y_j \in Y$ run protocol `find-matching`(i,j).

`find-matching`(i,j)

1. The server prepares t–out–of–T secret shares $[\overline{s}_1, \overline{s}_2, \dots \overline{s}_T]$ with secret $0^k || Y_j$, where k is the security parameter.
2. For every letter y_j^w in Y_j, the server computes: $v_w = ((\{x_i^w\}_{pk} -_h \{y_j^w\}_{pk}) \cdot_h r) + \{\overline{s}_w\}_{pk}$ which equals $\{((x_i^w - y_j^w) \cdot r + \overline{s}_w)\}_{pk}$, where r is always a fresh, random value from the domain of plaintext.
3. The server sends $[v_1, v_2, \dots v_T]$ to the client.
4. The client decrypts the values and checks if it is possible to reconstruct the secret $0^k || z$ from them. In order to do that, it needs to try all possible combinations of t among the T decrypted (potential) shares. If it is possible and z matches X_i then he adds z to his output set.

5.1.1 Correctness of protocol 3

In this protocol the client encrypts all of his words and sends results to the server. Then participants for every couple of words (X_i, Y_j) run a subroutine `find-matching`. In the subroutine the server divides his words into T shares (with the threshold t) and for every letter in Y_j calculates $v_w = \{((x_i^w - y_j^w) \cdot r + \overline{s}_w)\}_{pk}$. If $x_i^w = y_j^w$ then the client receives the correct share, otherwise a random value. However at this step the client cannot distinguish in which situation he is (he cannot distinguish a random value from the correct share). Then the client checks if he can reconstruct secret using any combination of t out of the T elements $\{D_{sk}(v_w) | 1 \leq w \leq T\}$. He recognizes the secret by the 0^k prefix, and similarity with one of the words from his set. If he has less than t correct secret shares then he cannot recover the secret and the retrieved data looks random to him (it follows from the security of the secret sharing scheme). Hence all required elements from Y appear in the client's output. The probability that some incorrect element is in the output set is negligible.

5.1.2 Security of protocol 3

The client input data is secure because all of the data received by the server is encrypted (using semantically secure cryptosystem). Hence the server cannot distinguish between different client inputs.

Privacy of the server is protected because the client receives correct secret shares of some $Y_j \in Y$ if and only if there is an element $X_i \in X$ such that $X_i \approx_t Y_j$. In this situation the client has at least t correct secret shares an he can reconstruct the secret $0^k||Y_j$. If there is no element in X to which Y_j is similar then the client receives n_S independent groups of shares, which has no group with at least t correct shares. Hence from any of these groups he cannot retrieve any secret. The probability that a random value from R is a correct share is negligible (with respect to security parameter k). Therefore the probability that the client can recover an illicit secret is negligible.

5.1.3 Complexity

Messages being sent in this protocol are encryptions of plaintext from the domain that contains D^T enlarged by k bits (where k is a security parameter). Optimization from Section 4.4 can be applied in a straightforward way to this protocol.

In step 2a the client sends encryptions of all letters of all his words to the server. This is $O(n_C \cdot T)$ messages. The subroutine `find-matching` is called $n_C \cdot n_S$ times. In this subroutine the server, in step 3, sends $O(T)$ ciphertexts. Hence, in total there are $O(n_C \cdot n_S \cdot T)$ messages sent in this protocol.

The main part of the server time complexity is preparing $n_S \cdot n_C$ times the T secret shares. Producing T secret shares can be done efficiently. Hence the time complexity of the server is reasonable.

The crucial part for the time complexity of the client is step 4 (that is performed once in every subroutine call). In this step the client verifies if he can reconstruct secret Y_j. Using the scheme from Section 2.3 checking if in a group of T potential shares there are t real shares costs $\binom{T}{t}$ reconstructions (and one reconstruction can be done efficiently). Hence the number of reconstructions is in the order of $O(n_S \cdot n_C \cdot \binom{T}{t})$. That is a big drawback of this protocol.

5.2 Improved protocol

The improved protocol is presented in Figure 4. For simplicity we assume here that in the server's set there are no matching elements:

$$\forall_{Y_i, Y_j \in Y} (i \neq j) \Rightarrow Y_i \not\approx_t Y_j$$

Later we describe which modification to the protocol is necessary to allow all of the possible input data in the protocol.

This protocol consists of a polynomial and a ticket phase. Aim of the polynomial phase is to provide to the client n lists of groups of encryptions of secret shares, from which he should be able to recover his output set (it is possible because of the simplification assumption). However, if he receives these shares in the plaintext he can abuse the protocol by gaining illicit information. Because of that there is a ticket phase whose aim is to protect the server privacy. This phase makes lists of groups of secret shares independent and therefore the client cannot mix shares from different groups to abuse the protocol.

5.2.1 Correctness of protocol 4

The first important issue appears in step 2 of the polynomial phase. Here the server prepares n groups of shares $[\overline{s}_{i,1}, \overline{s}_{i,2}, \ldots \overline{s}_{i,T}]$. From the ith group he can recover Y_i. During the creation of these shares the server uses the rule: if $y_i^w = y_m^w$ then $\overline{s}_{i,w} = \overline{s}_{m,w}$. This rule is necessary because later these shares are encoded as polynomials. It is possible to create these groups with such secrets

Figure 4: Improved protocol solving FPM problem

Polynomial Phase:

1. **The server** prepares sk, pk and $parameters$ for a semantically secure, additively homomorphic cryptosystem and sends pk and $parameters$ to the client.
2. For all $Y_i \in Y$, the server prepares t–out–of–T secret shares $[\overline{s}_{i,1}, \overline{s}_{i,2}, \ldots \overline{s}_{i,T}]$ with the secret $0^k||Y_i$, where k is the security parameter.
   If $y_i^w = y_m^w$ then $\overline{s}_{i,w} = \overline{s}_{m,w}$.
3. The server prepares T polynomials (for $w = 1$ to T) of degree n :
   (a) $((P_w(y_1^w) = \overline{s}_{1,w}) \cap (P_w(y_2^w) = \overline{s}_{2,w}) \cap \ldots (P_w(y_n^w) = \overline{s}_{n,w}))$
   (b) The server encrypts each polynomial $\{P_w\}_{pk}$ and sends it to the client.
4. The client evaluates T polynomials (for $w = 1$ to T) on each letter of each word (for $i = 1$ to n): $\{v_i^w\}_{pk} = \{P_w(x_i^w)\}_{pk}$.
   If $x_i^w = y_m^w$ then $v_i^w = \overline{s}_{m,w}$.
5. The client blinds the result v_i^w and sends them to the server: $\{v_i^w + r_i^w\}_{pk}$.

Ticket Phase:

6. For $i = 1$ to n, the server prepares t–out–of–T secret shares $[\overline{\tau}_{i,1}, \overline{\tau}_{i,2}, \ldots \overline{\tau}_{i,T}]$ with secret 0.
7. For $i = 1$ to n and for $w = 1$ to T, the server decrypts the received messages $D_{sk}(\{v_i^w + r_i^w\}_{pk})$ and sends $(v_i^w + r_i^w + \tau_i^w)$ to the client.
8. The client unblinds them (by subtracting r_i^w) and achieves q_i^w.
   If $x_i^w = y_m^w$ then $q_i^w = \overline{s}_{m,w} + \overline{\tau}_{i,w}$.
9. For $i = 1$ to n, the client checks if it is possible to reconstruct the secret $0^k||z$ from $[q_i^1, q_i^2, \ldots q_i^T]$. If it is possible and z is similar to any $X_i \in X$ then he adds z to his output set.

in the given manner because of the simplification assumption (otherwise for similar elements from set Y it would be impossible to choose different secrets for different groups of shares). After that, in step 3, the server creates T polynomials of degree n in such a way that evaluating the polynomial on a corresponding letter from some word from Y results in a corresponding secret share. Later he sends the polynomials to the client. The client evaluates the polynomials on his words and achieves $\{v_i^w\}_{pk}$ (where the following property holds: if $x_i^w = y_m^w$ then $v_i^w = \overline{s}_{m,w}$). After the ticket phase the client receives $q_i^w = v_i^w + \overline{\tau}_{i,w}$, where $[\overline{\tau}_{i,1}, \overline{\tau}_{i,2}, \ldots \overline{\tau}_{i,T}]$ are secret shares with the secret 0. Hence the client receives the group: $[v_i^1 + \overline{\tau}_{i,1}, v_i^2 + \overline{\tau}_{i,2}, \ldots v_i^T + \overline{\tau}_{i,T}]$, where if $x_i^w = y_m^w$ (for some $Y_m \in Y$) then $v_i^w = \overline{s}_{m,w}$. Therefore, by linear property of LSS, if v_i^w is a correct secret share then $v_i^1 + \overline{\tau}_{i,1}$ is also a correct secret share. Moreover, if in the group there are enough corresponding secret shares then the secret that could be recovered from them is $0^k||Y_m$ (because the secret of τ secret shares is 0). Hence, in step 9 the client recovers all of the secrets that he has corresponding shares.

5.2.2 Security of protocol 4

The client's privacy of the input data is secure because all of the data received by the server (in step 5 of the polynomial phase) is of the form: $v_i^w + r_i^w$, where r_i^w is a random value from the domain of the plaintext. Hence the server cannot distinguish between different client input.

The privacy of the server is protected because the client receives correct secret shares of some $Y_j \in Y$ if and only if there is an element $X_i \in X$ such that $X_i \approx_t Y_j$. Everything that client receives in the polynomial phase is encrypted so there is no leakage of information. He receives information in plaintext in step 7 of ticket phase 7. In this situation the client has at least t correct secret shares and he can reconstruct the secret $0^k||Y_j$.

If there is no such an element in X to which Y_j is similar then the client receives in every group of potential shares no more than t shares: $\overline{\tau}_{i,w} + \overline{s}_{j,w}$ (i is an index of the received group of potential shares). The client cannot reconstruct Y_j for any group separately (secret sharing assumption). τ values for every group of shares are different and make every received group of shares independent. The probability that a random value from R is a correct share is negligible (with respect to security parameter k). Therefore the probability that the client can recover illicit information is negligible.

5.2.3 Removing the simplification assumption

Steps that have to be changed to achieve independence of the simplification assumption are 2, 6, and 9. The following modifications should be made:

step 2 Instead of t–out–of–T secrets sharing of $\overline{s}$, $(t+T)$–out–of–$(2 \cdot T)$ secret scheme should be generated. Values $[\overline{s}_{i,T+1}, \overline{s}_{i,T+2}, \ldots \overline{s}_{i,2\cdot T}]$ should be sent in the plaintext to the client.

step 6 Instead of t–out–of–T secrets sharing of $\overline{\tau}$, $(t+T)$–out–of–$(2 \cdot T)$ secret scheme should be generated. Values $[\overline{\tau}_{i,T+1}, \overline{\tau}_{i,T+2}, \ldots \overline{\tau}_{i,2\cdot T}]$ should be sent in the plaintext to the client.

step 9 For $i = 1$ to n and $j = 1$ to n, the client checks if it is possible to reconstruct the secret $0^k||z$ from

$$[q_i^1,\ q_i^2,\ \ldots\ q_i^T,\quad \overline{s}_{j,T+1} + \overline{\tau}_{i,T+1},\ \overline{s}_{j,T+2} + \overline{\tau}_{i,T+2},\ \ldots\ \overline{s}_{j,2\cdot T} + \overline{\tau}_{i,2\cdot T}]$$

If it is possible and z is similar to any $X_i \in X$ then he adds z to his output set.

In this setting it is possible to perform step 2 even if some words from the server set are similar. This is achieved by making threshold of the sharing scheme big enough and sending additional shares in the plaintext. These additional shares are needed to prepare different secrets even if more than t shares are equal. These modification enlarge only the complexity of step 9 by factor n (in O sense).

5.2.4 Complexity

Here we discuss the complexity of protocol 4 (without the simplification assumption). Messages being sent in this protocol are encryptions of plaintext from the domain that contains D^T enlarged by k bits (where k is a security parameter). Optimization from Section 4.4 can be applied to this protocol in a straightforward way.

In step 3 the server sends encryptions of T polynomials of degree n, i.e., this is $O(n{\cdot}T)$ messages. For every received polynomial the client computes n values and sends them encrypted to the server (again $O(n \cdot T)$ messages). In the ticket phase the server answers to every received message with sending one non-encrypted value. Hence in whole protocol there are $O(n \cdot T)$ messages sent.

The main part of the server time complexity is preparing $2{\cdot}n$ times $2{\cdot}T$ secret shares. Producing $2 \cdot T$ secret shares can be done efficiently. Therefore time complexity of the server is reasonable.

The crucial part for the time complexity of the client is step 9 (this step is performed n^2 times). In this step the client verifies if he can reconstruct the secret Y_j. Using scheme from Section 2.3 checking if in a group of T potential shares there are t real shares costs $\binom{T}{t}$ reconstructions (and one reconstruction can be done efficiently). Hence in the time complexity of the client there is a factor $O(n^2 \cdot \binom{T}{t})$. That is a big drawback of this protocol.

6 Summary and Future Work

In this paper we have shown few protocols solving FPM problem. The most efficient one works in a linear bit complexity with respect to a size of the input data. However we cannot call this protocol really efficient, because of the slow time complexity of the client. Currently, we are researching to speed up the time complexity of the client by using error correcting codes. If we succeed in that we are planning to implement this protocol and make efficiency tests.

References

[BDS+03] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana Smetters, Jessica Staddon, and Hao-Chi Wong. Secret handshakes from pairing-based key agreements. In *24th IEEE Symposium on Security and Privacy*, Oakland, CA, May 2003.

[BST01] Fabrice Boudot, Berry Schoenmakers, and Jacques Traoré. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111(1–2):23–36, 2001.

[CJT04] Claude Castelluccia, Stanislaw Jarecki, and Gene Tsudik. Secret handshakes from ca-oblivious encryption, 2004.

[DA00] Kevi Du and Mike Atallah. Protocols for secure remote database access with approximate matching, 2000.

[FIM+01] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. *Lecture Notes in Computer Science*, 2076:927+, 2001.

[FNP04] Michael Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology — EUROCRYPT 2004.*, 2004.

[FNW96] Ronald Fagin, Moni Naor, and Peter Winkler. Comparing information without leaking it. *Communications of the ACM*, 39(5):77–85, 1996.

[GLLM04] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikainen. On private scalar product computation for privacy-preserving data mining. *Information Security and Cryptology - ICISC*, 2004.

[Gol02] Oded Goldreich. *Secure multi-party computation.* Cambridge University Press, 2002.

[IW06] Piotr Indyk and David Woodruff. Polylogarithmic private approximations and efficient matching. In *TCC 2006*, volume 3876 of LNCS, pages 245–264, 2006.

[KS05] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Advances in Cryptology — CRYPTO 2005.*, 2005.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. *Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 245–254, May 1–4 1999.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT 1999.*, pages 223–238, May 1999.

[Sha79] Adi Shamir. How to share a secret. In *Communications of the ACM, vol. 22, n.11*, pages 612–613, November 1979.

A Protocol solving private fuzzy matching problem based on computing hamming distance

A technique of computing an encrypted hamming distance to solve FPM problem is used in [IW06]. However, generic 2-party computations together with oblivious transfer are used in the protocol from that paper and that makes that solution less practical. We show two solutions (that use calculation of encrypted hamming distance): one that is simple and efficient for small domains and the second one that uses oblivious transfer. Oblivious transfer is described for example in [NP99]. Both solutions are shown in Figure 5. The difference between them is the implementation of subroutine `obtain-letters`.

Figure 5: Protocol solving private fuzzy matching problem using Hamming distance

Domain remark: like in protocol 1.

1. The client prepares sk, pk and $parameters$ for a semantically secure, additively homomorphic cryptosystem and sends pk and $parameters$ to the client.
2. Run subroutine `obtain-letters`.
   During this subroutine the server has obtained the following function:
   $$f(w,i,j) = \begin{cases} \{0\}_{pk}, & \text{for } x_i^w = y_j^w \\ \{1\}_{pk}, & \text{for } x_i^w \neq y_j^w \end{cases}$$
   where $w \in \{1, \dots T\}$ and $i, j \in \{1, \dots n\}$
3. For each $X_i \in X$ and $Y_j \in Y$ run protocol `find-matching`(i,j).

`find-matching`(i,j)

1. The server computes:
   $$\{\Delta(X_i, Y_j)\}_{pk} = \{\sum_{w=1}^{T} f(i,j,w)\}_{pk}$$
2. For $l = 0$ to $T - t$ the server sends $\{(\Delta(X_i, Y_j) - l) \cdot r + (0^k || Y_j))\}_{pk}$ to the client.

A.1 Correctness and Security of protocol 5

Assuming that in subroutine `obtain-letters` function f is set properly and security is maintained, then protocol 5 calculates a correct output. This follows from two facts:

- $\{\Delta(X_i, Y_j)\}_{pk}$ is calculated properly.
- If $X_i \approx_t Y_j$ then in step 2 subroutine `find-matching` $\Delta(X_i, Y_j) \in \{0 \dots T - t\}$ and therefore $\{0^k || Y_j\}_{pk}$ is sent to the client.

Formal proof of correctness of this protocol looks similar to the one of protocol 3.

Argument for privacy of the client is the same as for protocol 3.

Privacy of the server is protected because in step 2 of subroutine `find-matching` if $X_i \not\approx_t Y_j$ then $\Delta(X_i, Y_j) \notin \{0 \dots T - t\}$ and therefore all values received by the client look random to him.

A.2 Subroutines `obtain-letters`

Subroutine `v1` works properly because of the way vectors d_i^w are defined. All of the values received by the server are encrypted, hence security is maintained.

In subroutine `v2` the server wants to achieve encryption of $d_i^w(Y_j^w)$. In step 2 it receives $temp = d_i^w(Y_j^w) \oplus b_{i,j}^w$ and $\{b_{i,j}^w\}_{pk}$. The server wants to achieve: $\{temp \oplus b_{i,j}^w\}_{pk} = \{d_i^w(Y_j^w)\}_{pk}$. If $temp = d_i^w(Y_j^w) \oplus b_{i,j}^w = 0$ then $\{d_i^w(Y_j^w)\}_{pk} = \{b_{i,j}^w\}_{pk}$ and the server achieved the desired value. Otherwise, if $temp = 1$ then the following property holds:

$$\{(b_{i,j}^w + temp) - 2 \cdot (b_{i,j}^w \cdot temp)\}_{pk} = \{(b_{i,j}^w \oplus temp)\}_{pk} = \{d_i^w(Y_j^w)\}_{pk}$$

All of the values received by the server in this subroutine are encrypted or blinded by random values $b_{i,j}^w$. Hence, the security of this subroutine is maintained.

A.3 Conclusions

These protocols are less efficient than protocols presented in the main part of this paper. They strongly depend on the size of D and also contain factor n^2T in a bit complexity. Moreover, at the moment, we do not foresee a way how to improve them.

However, the protocols are interesting because they do not use generic 2-party computation. Furthermore techniques used there are interesting, especially subroutine `obtain-letters-v2` and the technique for obtaining encryption of a single bit using only one oblivious transfer.

Figure 6: Subroutines `obtain-letters` `-v1` and `-v2`

`obtain-letters-v1`

1. The client generates vectors: d_i^w: $[0..|D|-1]$ (where $i \in \{1, \dots n_C\}$ and $w \in \{1, \dots T\}$ such that:
   $d_i^w(v) = 1$ if $v = X_i^j$, and $d_i^w(v) = 0$ otherwise.
2. The client sends all $E(d_i^j(v))$ to the server.
3. The server sets function f by: $f(w, i, j) = d_i^j(w)$.

`obtain-letters-v2`

1. The client generates vectors: d_i^w: $[0..|D|-1]$ (where $i \in \{1, \dots n_C\}$ and $w \in \{1, \dots T\}$ such that:
   $d_i^w(v) = 1$ if $v = X_i^j$, and $d_i^w(v) = 0$ otherwise.
2. Function f is set in the following way (for all $i, j \in \{1, \dots n\}$ and $w \in \{1, \dots T\}$):
   (a) The server and client performs oblivious transfer in following manner:
   $temp := oblivious\_transfer(Y_j^w, h_{i,j}^w)$ where:
   - Y_j^w represents the index that the server is asking for.
   - $h_{i,j}^w$ is a vector $[0..|D|-1]$ defined in the following way:
     $h_{i,j}^w = d_i^w \oplus b_{i,j}^w = [d_i^w(0) \oplus b_{i,j}^w, d_i^w(1) \oplus b_{i,j}^w, \dots d_i^w(|D|-1) \oplus b_{i,j}^w]$
     This vector contains an element that is interesting for the server element and is owned by the client.
   - $b_{i,j}^w$ is a random bit.

   (b) The client sends $\{b_{i,j}^w\}_{pk}$ to the server.
   (c) $f(w, i, j) =$

$$\begin{cases} \{b_{i,j}^w\}_{pk}, & \text{for } temp = 0 \\ \{(b_{i,j}^w + temp) - 2 \cdot (b_{i,j}^w \cdot temp)\}_{pk}, & \text{for } temp = 1 \end{cases}$$