Fuzzy Private Matching (Extended Abstract)

Łukasz Chmielewski[*] Jaap-Henk Hoepman[*†]
[*]Security of Systems (SoS) group
Institute for Computing and Information Sciences, Radboud University Nijmegen
{lukaszc,jhh}@cs.ru.nl
[†]TNO Information and Communication Technology
P.O. Box 1416, 9701 BK, Groningen, The Netherlands
jaap-henk.hoepman@tno.nl

***Abstract*—In the private matching problem, a client and a server each hold a set of n input elements. The client wants to privately compute the intersection of these two sets: he learns which elements he has in common with the server (and nothing more), while the server gains 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.* [1] is incorrect. Subsequently we present two fuzzy private matching protocols. The first, simple, protocol has bit message complexity $O(n\binom{T}{t}(T\log|D|+k))$. The second, improved, protocol has a much better bit message complexity of $O(nT(\log|D|+k))$, but here the client incurs a $O(n)$ factor time complexity. Additionally, we present protocols based on the computation of the Hamming distance and on oblivious transfer, that have different, sometimes more efficient, performance characteristics.

***Index Terms*—fuzzy matching, secure 2-party computation, secret sharing**

I. Introduction

In the private matching problem [1], a client and a server each hold a set of elements as their input. The size of the set is 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 the server (and nothing more), while the server obtains no information at all.

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 similar, but not necessarily equal. Such a private matching is called *fuzzy*, and was introduced by Freedman *et al.* [1]. Elements are called similar (or matching) in this context if they match on t out of T letters at the right locations.

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

All known solutions for fuzzy private matching, as well as our own protocols, work in a semi-honest environment. In this environment participants do not deviate from their protocol, but may use any (additional) information they obtain to their own advantage.

Freedman *et al.* [1] 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 III): 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 or FPM for short). The first, simple, protocol has time complexity $O(n\binom{T}{t})$ and bit message complexity $O(n\binom{T}{t}(T\log|D|+k))$ (protocol 3). The second protocol is based on linear secret sharing and has a much better bit message complexity $O(nT(\log|D|+k))$ (protocol 5). Here the client incurs a $O(n^2\binom{T}{t})$ time complexity penalty. Note that this is only a factor n worse than the previous protocol. We also present a simpler version of protocol 5 (protocol 4) to explain the techniques used incrementally. This protocol has a slightly worse bit message complexity.

Note that, contrary to intuition, fuzzy extractors and secure sketches ([2]) cannot be used to solve fuzzy private matching problem.

Indyk and Woodruff [3] present another approach for solving fuzzy private matching, using the computation of the Hamming distance together with generic techniques like secure 2-party computations and oblivious transfer. Generic multi-party computation and oblivious transfer are considered not to be efficient techniques. Therefore, based on the protocol from [3], we design protocols based on computation the Hamming distance that do not use secure 2-party computation. One protocol is efficient for small domains of letters (protocol 6 version 1) and the second protocol uses oblivious transfer (protocol 6 version 2). The major drawback of the first protocol is a strong dependence on the size of the domain of letters. The main weakness of the second protocol is its high complexity

	Bit Complexity ($\tilde{O}$)	**Time Complexity**[1]	**Bit Complexity** (O)
[1] (corrected), Fig.3 protocol	$n\binom{T}{t}$	$O(n\binom{T}{t})$	$n\binom{T}{t}(T\log\lvert D\rvert+k)$
SFE protocol	n^2T	$\tilde{O}(n^2T)$	$n^2Tk\log\lvert D\rvert$
[3]	nT^2+n^2	$\tilde{O}(nT^2+n^2)$	— [2]
Fig.4 protocol	n^2T	$O(n^2T\binom{T}{t})$	$n^2T(\log\lvert D\rvert+k)$
Fig.5 protocol	nT	$O(n^2T\binom{T}{t})$	$nT(\log\lvert D\rvert+k)$
Fig.6 protocol v1[3]	$\lvert D\rvert nT+n^2(T-t)$	$O(\lvert D\rvert nT+n^2(T-t))$	$\lvert D\rvert nTk+n^2(T-t)(T\log\lvert D\rvert+k)$
Fig.6 protocol v2[4]	n^2T	n^2T *oblivious transfer* calls	n^2T *oblivious transfer* calls

[1] For the sake of simplicity time complexities are given roughly in numbers of efficient operations (e.g., secret sharing's reconstructions, encryptions, polynomial's evaluations etc.); we also report here only the complexity of the slowest participant

[2] the authors of the paper do not give exact complexity in the O notation.

[3] protocol with subroutine from first paragraph of section VI-A.

[4] protocol with subroutine `equality-matrix` from Figure 7.

Fig. 1. Results overview

– in the protocol there are $n^2 \cdot T$ oblivious transfer calls. We present these protocols mainly to show that other approaches to solve the fuzzy private matching problem exist as well.

We compare our protocols to existing solutions using several complexity measures in Table 1. One of these complexity measures is the $\tilde{O}$ notation used for the bit message complexity in [3]. This notation is defined as follows. For functions f and g, we write $f = \tilde{O}(g)$ if $f(n,k) = O\left(g(n,k)\ \log^{O(1)}(n)\cdot \text{poly}(k)\ \right)$, where k is the security parameter. This notation hides certain factors like a strong dependence on the security parameter k (e.g. k^3), and is therefore less accurate than the standard big-O notation. We prefer this measure for the plain message complexity, where we restrict the bit size of the messages to be linear in k.

Related work can be traced back to private equality testing [4], [5], [1], [6] 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 [1], [6], [7] (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 the so called secret handshaking protocols [8], [9], [10]. 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 issues have also been considered for the approximation of a function f among vectors owned by several parties. The function f may be Euclidean distance ([11], [12], [3]), set difference ([1]), Hamming distance ([11], [3]), or scalar product (reviewed in [13]).

Our paper is structured as follows. We formally define the fuzzy private matching problem in Section II, and introduce our system model, some additional notation, and primitives there as well. Then in Section III we present the solution from [1] for 2-out-of-3 fuzzy private matching and show where it breaks down. Section IV contains our first protocol for t-out-of-T fuzzy private matching that uses techniques similar to the ones used in [1]. Then we present our second protocol based on linear secret sharing in Section V. Finally, Section VI presents two protocols based on the computation of a Hamming distance. All our protocols assume a semi-honest environment (see Section II-B).

II. 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.

A. Fuzzy Private Matching Problem Definition

Let a client and a server each own a set of words. A fuzzy private matching protocol 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 each word $X = x^1 \ldots x^T$ in these sets consist of T letters x^i from a domain D. Let $X = x^1 \ldots x^T$ and $Y = y^1 \ldots y^T$. We define $X \approx_t Y$ (X and Y 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. The client input is the set $X = \{X_1, \ldots X_{n_C}\}$ of n_C words of length T, while the server's input is defined as $Y = \{Y_1, \ldots Y_{n_S}\}$ of n_S words of length T. Both the client and the server have also in their inputs n_C, n_S, T and t. The output of the client is the 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. The server's output 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 a priori known to the other party (so the protocol does not have to prevent the other party to learn the size of the set).

B. Adversary Models

We prove correctness of our protocols only against computationally bounded (with respect to a security parameter k) and semi-honest adversary, meaning the the parties follow the protocol but may keep message histories in an attempt to learn more than is prescribed. Here we provide the intuition and the informal notion of this model, the reader is referred to [14] for full definitions. To simplify matters we only consider the case of only two participants, the client and the server.

We have chosen the semi-honest model for a few reasons. First of all, there had not been made any "really" efficient solution for FPM problem in any model. Secondly, our protocols seem to be secure against malicious clients and the only possible attacks are on the correctness of the protocols by malicious servers. Moreover in [15], [16], [17], it is shown how to transform a semi-honest protocol into a protocol secure in the malicious model. Further, [17] does this at a communication blowup of at most a small factor of $poly(k)$. Therefore, we assume parties are semi-honest in the remainder of the paper (however we are aware that the mentioned generic transformations are not too efficient).

We leave improving protocols to work efficiently in malicious environment and proofs that the protocols from this paper are secure against malicious clients for future work.

In the model with a semi-honest adversary, both parties are assumed to act accordingly to 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 (the client) learns the output. Following [1] 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.

C. Additively Homomorphic Cryptosystem

In all our protocols we use a semantically secure, additively homomorphic public-key cryptosystem, e.g., Paillier's cryptosystem [18]. Let $\{\cdot\}_K$ denote the encryption function with the public key K. The homomorphic cryptosystem supports the following two operations, which can be performed without the knowledge of the private key.

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

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.

1) Remark: The domain R of the plaintext of the homomorphic cryptosystem in all of our protocols (unless specified differently) is defined as follows: R should be larger than D^T (or in some protocols D) and a uniformly random element from R should be in D^T (or D) with negligible probability. This property can be satisfied by representing an element $a \in D^T$ (or in some protocols $a \in D$) by $r_a = 0^k || a$ in R. The domain R should be a field (e.g., $\mathbb{Z}_q$ for some prime q).

2) Operations on encrypted polynomials: We represent any polynomial p of degree n (on some ring) as the ordered list of its coefficients: $[\alpha_0, \alpha_1, \ldots \alpha_n]$. We denote the encryption of a polynomial p by $\{p\}_K$ and define it to be the list of encryptions of its coefficients: $[\{\alpha_0\}_K, \{\alpha_1\}_K, \ldots \{\alpha_n\}_K]$.

Many operations can be performed on such encrypted polynomials like: addition of two encrypted polynomials or multiplication of an encrypted and a plain polynomial. We use the following property: given an encryption of a polynomial $\{p\}_K$ and some x one can efficiently compute a value $\{p(x)\}_K$. This follows from the properties of the homomorphic encryption scheme:

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

D. Linear Secret Sharing

Some of our protocols use t-out-of-T secret sharing. The secret $\overline{s}$ is split into T secret shares $\overline{s}^i$, such that any combination of at least t such shares can be used to reconstruct $\overline{s}$. Combining less than t individual shares gives no information whatsoever about the secret.

A Linear t-out-of-T Secret Sharing (LSS) scheme is a secret sharing scheme with 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 indices, using $\overline{s}^i + \overline{r}^i$ one can reconstruct the sum of the secrets $\overline{s} + \overline{r}$. One such LSS scheme is Shamir's original secret sharing scheme [19].

III. The Original FPM Protocol

Freedman *et al.* [1] proposed a fuzzy private matching protocol for the case where $T = 3$ and $t = 2$ (see Figure 2). Unfortunately, their protocol is incorrect.

1) The idea behind, and the problem of the protocol from Figure 2: 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 $\{r' \cdot (P_2(y_j^2) - P_3(y_j^3)) + Y_j\}_K$ sent back by the server simplifies to $\{Y_j\}_K$ (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.

1) The client chooses a private key sk, a public key K and *parameters* for the additively homomorphic encryption scheme and sends K and the *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 over R (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.
3) For each Y_j (such that $1 \leq j \leq n_S$), the server responds to the client: $\{r \cdot (P_1(y_j^1) - P_2(y_j^2)) + Y_j\}_K$, $\{r' \cdot (P_2(y_j^2) - P_3(y_j^3)) + Y_j\}_K$, $\{r'' \cdot (P_1(y_j^1) - P_3(y_j^3)) + Y_j\}_K$, where r, r', r'' are fresh random values in R. This uses the properties of the homomorphic encryption scheme including the encrypted polynomials explained in Section II-C2.
4) If the client receives an encryption of an encoding of Y_i, which is similar to any word from his set X, then he adds it to the output set.

Fig. 2. Original FPM protocol

1) The client chooses a private key sk, a public key K and *parameters* for the additively homomorphic encryption scheme and sends K and the *parameters* to the server.
2) For every combination σ of t out of T indices the client:
   a) constructs a polynomial:
   $P_\sigma(x) = (x - \sigma(X_1)) \cdot (x - \sigma(X_2)) \cdots (x - \sigma(X_{n_C}))$ of degree n_C with domain D^T and range R.
   b) sends $\{P_\sigma\}_K$ (the encrypted polynomial) to the server.
3) For every $Y_i \in Y$, $1 \leq i \leq n_S$, and every received polynomial $\{P_\sigma\}_K$ (corresponding to the combination σ) the server:
   a) evaluates polynomial $\{P_\sigma\}_K$ at the point $\sigma(Y_i)$ to compute $\{w_i^\sigma\}_K = \{r * P_\sigma(\sigma(Y_i)) + Y_i\}_K$, where $r \in R$ is always a fresh random value.
   b) sends $\{w_i^\sigma\}_K$ to the client.
4) The client decrypts all received messages. If for such a decryption $w_i^\sigma \approx_t X_j$ for any $X_j \in X$, then he adds w_i^σ to the output set.

Fig. 3. Polynomial Based Protocol solving FPM problem

There is however a problem with this approach. Consider the following input data. The input of the client is $\{[1,2,3]$, $[1,4,5]\}$, while the input of the server is $\{[5,4,3]\}$. Then in step 2c of the protocol, the polynomials are defined (by the client) in the following way: $P_1(1) = r_1 \cap P_1(1) = r_2$, $P_2(2) = r_1 \cap P_2(4) = r_2$ and $P_3(3) = r_1 \cap P_3(5) = r_2$. 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 run into another one. Among other things, the server computes $\{r' \cdot (P_2(y_i^2) - P_3(y_i^3)) + Y_i\}_K$, which, in this particular case equals $\{r' \cdot (P_2(4) - P_3(3)) + [5,4,3]\}_K$. This equals $\{r' \cdot (r_2 - r_1) + [5,4,3]\}_K$, which by equality of r_1 and r_2 reduces to $\{[5,4,3]\}_K$. 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: if a semi-honest client happens to own a set of tuples with a property similar to the counterexample above, it learns a tuple of the server.

IV. A Polynomial Based Protocol

The protocol of the previous section can be fixed, but in a slightly more elaborate way. Our solution works for any T and t, and is presented in Figure 3. In the protocol we use the following definition. Let σ be a combination of t different indices $\sigma_1, \sigma_2, \ldots, \sigma_t$ 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 indices in the combination). We now discuss the correctness, security and complexity of this protocol.

2) Correctness: 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. In fact, the roots of the polynomial P_σ are $\sigma(X_i)$ It is easy to see that if $X \approx_t Y$ then $\sigma(X) = \sigma(Y)$ for some combination σ. Hence, if $X_i \approx_t Y_j$ then $P_\sigma(\sigma(Y_j)) = 0$ for some P_σ received and evaluated in step 3a. When that happens, the encryption of Y_j is sent to the client. Later on, the client can recognize this value by the convention that values in D^T are represented in R using a 0^k prefix. Otherwise (if Y_j does not match with any element from X) all the values sent to the client contain a random blinding element r (and therefore their decryptions are in Y with negligible probability).

3) Security: The client's input data is secure because all the data received by the server are encrypted (using a semantically secure cryptosystem). Hence the server cannot distinguish between different client's inputs. The privacy of the server is protected because the client only learns about those elements from Y that are also in X, and because (by semi-honesty) it does not send specially constructed polynomials to cheat the server. If an element $y_i \in Y$ does not belong to X then a random value is sent by the server (see the correctness proof above).

4) Complexity: The messages being sent in this protocol are encryptions of plaintext from the domain R, i.e., $O(T \log |D| + k)$ bits. In step 2 the client sends $\binom{T}{t}$ polynomials of degree n_C (sending each coefficient separately). 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 sent. Therefore, the total bit complexity is $O((n_S + n_C) \cdot \binom{T}{t} \cdot (T \log |D| + k))$.

The time complexity is the same as the number of messages in protocol $O((n_S + n_C) \cdot \binom{T}{t})$.

V. Secret Sharing Based Protocols

The number of messages sent in the previous protocol is very large. Therefore, we now present two protocols solving the FPM problem based on linear secret sharing that trade a decrease in message complexity for an increase in time complexity. Both work in the model with a semi-honest adversary. First we describe the simple (but slow) protocol and later the faster, improved one. We present the simple version mainly to facilitate the understanding of the improved protocol.

A. A Simple Version of the Protocol

The simple protocol is presented in Figure 4. The idea behind the protocol is the following. The server encrypts all

1) The client generates sk, K and *parameters* for the additively homomorphic cryptosystem and sends K and the *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 $\{x_i^w\}_K$ to the server.
   b) For each $Y_j \in Y$, run the protocol `find-matching`(i,j).

`find-matching`(i,j):

1) The server generates sk_j and *parameters* for the symmetric cryptosystem and sends *parameters* to the client.
2) The server sends $\widehat{y_j} = E_{sk_j}(Y_j)$ to the client.
3) The server prepares t–out–of–T secret shares $[\overline{s}^1, \overline{s}^2, \ldots \overline{s}^T]$ with secret $0^k || sk_j$, where k is the security parameter.
4) For every letter y_j^w in Y_j, the server computes: $v_w = ((\{x_i^w\}_K -_h \{y_j^w\}_K) \cdot_h r) +_h \{\overline{s}^w\}_K$ which equals $\{((x_i^w - y_j^w) \cdot r + \overline{s}^w)\}_K$, where r is always a fresh, random value from the domain of plaintext.
5) The server sends $[v_1, v_2, \ldots v_T]$ to the client.
6) The client decrypts the values and checks whether it is possible to reconstruct the secret $0^k || z$ from them. In order to do that, he needs to try all possible combinations of t among the T decrypted (potential) shares. If it is possible and $Dec_z(\widehat{y_j}) \approx_t X_i$ then he adds $Dec_z(\widehat{y_j})$ to his output set.

Fig. 4. Simple secret sharing protocol solving FPM problem

its words Y_j using separate symmetric keys sk_j and sends the results to the client. The protocol then proceeds to reveal key sk_j to the client only if there is a word X_i such that $X_i \approx_t Y_j$.

Every word X_i of the client is matched with each word Y_j of the server one by one. To this end, the client first sends each letter of X_i to the server, encrypted to the public key of the server separately.

Upon reception of the encrypted letters for X_i, the server does the following for each word Y_j in his set (using the subroutine `find-matching`(i,j)). Firstly the server prepares secret key (sk_j for corresponding word Y_j) for the symmetric encryption scheme (e.g., AES), and sends the encrypted Y_j to the client. Then it prepares t-out-of-T random secret shares $\overline{s}^1, \ldots, \overline{s}^T$ such that $\overline{s} = 0^k || sk_j$. Share $\overline{s}^i$ is "attached" to the i-th letter of word Y_j, so to speak. Note that each time a new word X_i from the client is matched with Y_j, *fresh* secret shares are generated to avoid an attack similar to the one described in section III.

Using the homomorphic properties of the encryption scheme, the server then computes for each encrypted letter $\{x_i^w\}_K$ it received, the value $v_w = \{((x_i^w - y_j^w) \cdot r + \overline{s}^w)\}_K$ (using a fresh random value r each time, and encrypting y_j^w to the public key K). Note that $v_w = \{\overline{s}^w\}_K$ if and only if $x_i^w = y_j^w$.

Finally, the server sends $v_1, \ldots, v_T$ back to the client. The client decrypts these values, and if $X_i \approx_t Y_j$, then by the observation in the previous paragraph, among the decrypted values there are at least t shares $\overline{s}^w$ from which sk_j and therefore Y_j can be reconstructed.

Due to space constraints we skip the proofs of correctness and security of the protocol from Figure 4 (they can be found in the appendix).

1) Complexity: Two kinds of messages are sent in this protocol. Messages encrypted by homomorphic encryption scheme are from the domain $O(\log |D| + k)$ bits. The second kind of messages are the messages encrypted by the symmetric encryption scheme (they are sent in step 2 of the subroutine). They are encryptions of plaintext from the domain D^T.

The main impact on the message complexity of the protocol is the fact that the subroutine `find-matching` is called $n_C n_S$ times. In this subroutine, the server sends $O(T)$ ciphertexts in step 5 . Hence, in total $O(n_C n_S T)$ messages of size $O(\log |D| + k)$ and $O(n_S)$ messages of size $O(\log |D|^T + k)$ are sent in this protocol. Therefore, the bit complexity of the protocol is: $O(n_C n_S T(\log |D| + k) + n_S(\log |D|^T + k)) = O(n_C n_S T(\log |D| + k))$.

We see that by first encrypting the words stored by the server using symmetric keys, and later using the secret sharing mechanism to reveal these keys instead of the full words, changes the bit complexity from $O(T(\log |D|^T + k))$ to $O(T(\log |D| + k))$, removing a factor T.

The server prepares $n_S n_C$ times the T secret shares. Producing T secret shares can be done efficiently and therefore the time complexity of the server is reasonably low. The client (in step 6 for each subroutine call) verifies if he can reconstruct the secret Y_j. This verification costs $\binom{T}{t}$ reconstructions (and one reconstruction can be done efficiently). The number of reconstructions is in the order of $O(n_S n_C \binom{T}{t})$, which is the major drawback of this protocol.

B. An Improved Protocol

We can improve the message complexity by combining the idea of using secret sharing (protocol 4) with the idea of encoding all characters at position w using a polynomial P_w (protocol 2). The resulting protocol for FPM is presented in Figure 5. It consists of two phases: a polynomial phase, and a ticket phase.

The polynomial phase runs as follows. As in the previous protocol, words are first sent encrypted to the client, while the key sk_j is encoded using a secret sharing scheme such that when the client has a word matching on letter w, it obtains share $\overline{s_j}^w$.

However, we now encode the shares at letter position w using a polynomial P^w defined by

$$(P^w(y_1^w) = \overline{s_1}^w) \cap (P^w(y_2^w) = \overline{s_2}^w) \cap \ldots \cap (P^w(y_n^w) = \overline{s_n}^w)$$

(where, for technical reasons, at least random point is added to ensure privacy in the case $x_i^w \neq y_j^w$). This polynomial is sent to the client to allow him to recover share $\overline{s_i}^w$ for each letter $x_i^w = y_j^w$. In fact, it is sent encrypted to the client; more about this later.

We need to avoid the problem discussed in section III with the original FPM protocol. Observe that the above definition of P^w is only valid if we require that $\overline{s_i}^w = \overline{s_j}^w$ whenever $y_i^w = y_j^w$. This means that, as we proceed through to the list of words Y_j of the server constructing secret shares for key sk_j, we accumulate restrictions on the possible share values we can use. In the extreme case, for some word Y_j, T shares could already be fixed! If T was the total number of shares, then sk_j would be fixed and we would have the same leakage of information discussed in section III.

We solve this problem by adding an extra shares $\overline{s_j}^{T+1}, \ldots$ (that are in fact sent to the client in the clear!) and changing the parameters of the secret sharing scheme, as follows. We observe that if at most T shares can get fixed as described above, the best we can do is create a $(T+1)$-out-of-$(T+x)$ scheme. This ensures that an arbitrary sk_j can actually be encoded by the secret sharing scheme, even given T fixed shares. The x extra shares are given away "for free" to the client. Now to ensure that the client needs at least t letters that match word Y_j in order to be able to reconstruct sk_j form the shares it receives, we need $t = T + 1 - x$ i.e., $x = T + 1 - t$.

In other words, we use a $(T+1)$-out-of-$(2 \cdot T + 1 - t)$ secret sharing scheme where for each word Y_j

- the first T shares are encoded using polynomials $P^1, \ldots, P^T$, and
- the remaining $T + 1 - t$ shares are given the client in the clear.

If $X_i \approx_t Y_j$, then the client obtains at least t shares using the polynomials $P^1, \ldots, P^T$. Combined with the $T+1-t$ shares it got for free, it owns at least $T+1$ shares that allow it to reconstruct the secret. Note, however, that when it obtains the shares by evaluating the polynomial for the letters in X_i, it does not know to which Y_j these shares actually correspond. So in fact to actually try to reconstruct the secret, it needs to combine these shares with each group of free $T+1-t$ shares corresponding to Y_1 up to Y_n one by one.

This works, but it still leaves the leakage of information problem discussed in section III when several different words held by the client each match on some characters of a word Y_j held by the client, such that t shares for sk_j are released even though no single word of the client actually matches Y_j. This problem is solved in the ticket phase, as follows.

In fact, the polynomials sent by the server to the client are encrypted using the homomorphic encryption scheme. Therefore, when evaluating the polynomials for a word X_i, the client only obtains the *encrypted* shares corresponding to it. These are useless by themselves. The client needs the help of the server to decrypt these shares. In doing so, the server will enforce that the shares the client receives in the end actually correspond to a single word in the client set (and not a mix of shares obtained using letters from different words as in the attack described in the previous paragraph).

The server enforces this using so-called tickets (hence the name: ticket phase). Tickets are in fact $(T+1)$-out-of-$(2 \cdot T + 1 - t)$ random secret shares for the secret 0. The clients sends groups of encrypted shares (blinded by random values) that he got for every word X_i to the server. The server, for every group of shares received from the client, decrypts these shares and adds the tickets shares. The result is sent back to the client, who unblinds the result (subtracting the random value). Because of the linear property of the secret sharing scheme, the secret corresponding to the shares the client receives in the end (that are the sum of the original share and the ticket share) has not changed. But if the client tries to combine different shares obtained form different words, the shares of the tickets hidden within them no longer match and reconstruction of the secret is prevented.

Polynomial Phase:

1) **The server** prepares sk, K and *parameters* for the additively homomorphic cryptosystem and sends K and the *parameters* to the client.
2) For all $Y_j \in Y$, the server generates sk_j and *parameters* for the symmetric cryptosystem and sends *parameters* to the client. Later the server sends $\widehat{y_j} = E_{sk_j}(0^k || Y_j)$ to the client.
3) For all $Y_j \in Y$, the server prepares $[T+1]$–out–of–$[2 \cdot T - t + 1]$ secret shares $[\overline{s_j}^1, \overline{s_j}^2, \ldots \overline{s_j}^{2 \cdot T - t + 1}]$ with the secret $0^k || sk_j$, where k is the security parameter. If $y_j^w = y_m^w$ then $\overline{s_j}^w = \overline{s_m}^w$.
   The server sends $[\overline{s_j}^{T+1}, \ldots \overline{s_j}^{2 \cdot T - t + 1}]$ to the client.
4) The server prepares T polynomials (for $w = 1$ to T) of degree n :
   a) The polynomial is defined in the following way: $((P^w(y_1^w) = \overline{s_1}^w) \cap (P^w(y_2^w) = \overline{s_2}^w) \cap \ldots (P^w(y_n^w) = \overline{s_n}^w))$
      The number of points is increased to $n+1$ by adding random points (at least one random point is added).
   b) The server computes the coefficients of the polynomials and encrypts each polynomial $\{P^w\}_K$ and sends it to the client.
5) The client evaluates T polynomials (for $w = 1$ to T) on each letter of each word (for $i = 1$ to n): $\{v_i^w\}_K = \{P^w(x_i^w)\}_K$. If $x_i^w = y_m^w$ then $v_i^w = \overline{s_m}^w$.
6) The client blinds the results v_i^w with a random values r_i^w and sends them to the server: $\{v_i^w + r_i^w\}_K$.

Ticket Phase:

6) For $i = 1$ to n, the server prepares $[T+1]$–out–of–$[2 \cdot T - t + 1]$ secret shares $[\overline{\tau_i}^1, \overline{\tau_i}^2, \ldots \overline{\tau_i}^{2 \cdot T - t + 1}]$ with secret 0. Later he sends $[\overline{\tau_i}^{T+1}, \ldots \overline{\tau_i}^{2 \cdot T - t + 1}]$ to the client.
7) For $i = 1$ to n and for $w = 1$ to T, the server decrypts the received messages $D_{sk}(\{v_i^w + r_i^w\}_K)$ and sends $(v_i^w + r_i^w + \overline{\tau_i}^w)$ to the client.
8) The client unblinds them (by subtracting r_i^w) obtaining 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 and $j = 1$ to n, the client checks if it is possible to reconstruct the secret $0^k || z$ from: $[q_i^1,\ q_i^2,\ \ldots\ q_i^T, \overline{s_j}^{T+1} + \overline{\tau_i}^{T+1}, \overline{s_j}^{T+2} + \overline{\tau_i}^{T+2}, \ldots \overline{s_j}^{2 \cdot T - t + 1} + \overline{\tau_i}^{2 \cdot T - t + 1}]$.
   In order to do that, the client needs to try all possible combinations of t shares among the T decrypted q shares (the rest of the shares is the same during reconstructions). If it is possible and for any $\widehat{y_j}$, $Dec_z(\widehat{y_j}) = 0^k || a$, and a matches X_i then he adds a to his output set.

Fig. 5. Improved secret sharing protocol solving FPM problem

Due to space constraints we skip the proofs of correctness (that is essentially similar to the discussion above) of the protocol from Figure 5. This proof can be found in the appendix.

1) Security: The privacy of the client's input data is secure because all of the data received by the server (in step 6 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 inputs.

The privacy of the server is protected because the client receives correct secret shares of some sk_j (corresponding to $Y_j \in Y$) if and only if there is an element $X_i \in X$ such that $X_i \approx_t Y_j$. In the polynomial phase, the client receives encrypted polynomials and n groups with $T - t + 1$ shares ($[\overline{s_i}^{T+1}, \ldots \overline{s_i}^{[2 \cdot T - t + 1]}]$) of $[T+1]$–out–of–$[2 \cdot T - t + 1]$ secret sharing scheme. Hence, there is no leakage of information in the polynomial phase. The client receives information in plaintext in steps 6 and 7 of the ticket phase. In this situation, the client has at least $T+1$ correct secret shares during step 7 and he can reconstruct the secret $0^k || sk_m$ (and therefore, Y_m).

If there is no such element in X to which Y_j is similar, then the client receives no more than t shares in every group q_i of potential shares: $q_i^w = \overline{\tau_i}^w + \overline{s_j}^w$ (where i is an index of the received group of potential shares). The other values (for incorrect letters) include $P^w(y_j^w)$ that cannot be determined. It is caused by the fact that the client does not know enough points (degree of the polynomial is $n+1$ and the client can know only n points) defining the polynomial and at least one unknown point is random. This is exactly the situation like in a polynomial based secret sharing scheme when not enough shares are known. The client cannot reconstruct sk_j for any group separately (by the secret sharing assumption), because he has less than $T+1$ correct secret shares. Of all the shares, $(T-t+1)$ come from values that are sent in plaintext. For every group of shares, τ values are different and therefore make every received group of shares independent. The probability that a random value from R is a correct share is negligible (with respect to a security parameter k). Therefore, the probability that the client can recover illicit information is negligible.

2) Complexity: In step 2 the server sends n messages encrypted by the symmetric encryption scheme that are from the domain $O(\log |D|^T + k)$ (that is $O(n(T \log |D| + k))$ bits). Later in step 3 the server sends $O(nT)$ unencrypted messages from the domain $O(k + \log |D|)$ (that is $O(nT(\log |D| + k))$ bits). In step 4 the server sends encryptions of T polynomials of degree n. This totals to $O(nT(\log |D| + k))$ bits. For every received polynomial, the client computes n values and sends them encrypted to the server (again $O(nT(\log |D| + k))$ bits). In the ticket phase, in step 7, the server sends $O(nT)$ unencrypted messages, that is $O(nT(\log |D| + k))$ bits. Hence, the bit complexity of the entire protocol totals to: $O(nT(k + \log |D|) + n(k + \log |D|^T)) = O(nT(k + \log |D|))$.

The main part of the server time complexity is preparing $2n$ times $[T+1]$–out–of–$[2 \cdot T - t + 1]$ secret shares. Since producing $(2 \cdot T - t + 1)$ secret shares can be done efficiently, the time complexity of the server is reasonable. The crucial part for the time complexity of the client is step 9 (which is performed n^2 times). In this step the client checks whether he can reconstruct the secret Y_j. This verification costs $\binom{T}{t}$ reconstructions (and one reconstruction can be done efficiently). The total number of reconstructions is in the order of $O(n^2 \binom{T}{t})$, which is the major drawback of this protocol.

VI. Hamming Distance Based Protocol

In this section we present two protocols solving the FPM problem based on computing the encrypted Hamming distance: one that is simple and efficient for small domains and another that uses oblivious transfer. The difference between them is only the implementation of the subroutine `equality-matrix` (the frame of the protocol is the same for both of them). Firstly we describe the simple protocol and later the one using oblivious transfer.

A technique to compute the encrypted Hamming distance to solve the FPM problem has been introduced in [3]. However, the protocol in that paper uses generic 2-party computations together with oblivious transfer, making their approach less practical.

1) The client prepares sk, K and the *parameters* for the additively homomorphic cryptosystem and sends K and the *parameters* to the server.
2) Run subroutine `equality-matrix`. After this subroutine the server has obtained the following matrix:
$$f(w,i,j) = \begin{cases} \{0\}_K, & \text{for } x_i^w = y_j^w \\ \{1\}_K, & \text{for } x_i^w \neq y_j^w \end{cases},$$
where $w \in \{1, \ldots T\}$ and $i, j \in \{1, \ldots n\}$
3) For each $X_i \in X$ and $Y_j \in Y$:
   a) the server computes $\{\Delta(X_i, Y_j)\}_K = \{\sum_{w=1}^{T} f(i,j,w)\}_K$ and, for $\ell = 0$ to $T - t$, sends $\{(\Delta(X_i, Y_j) - \ell) \cdot r + (0^k || Y_j))\}_K$ to the client. Here r is always a fresh, random value.
   b) The client decrypts all $T - t$ messages and if any plaintext is in D^T and matches any word from X, then the client adds this plaintext to the output set.

Fig. 6. Hamming distance based protocol for the FPM problem

Our protocol (see Figure 6) works as follows. The server first obtains, using the subroutine `equality-matrix`, a 3-dimensional matrix $f(w,i,j)$ containing the encrypted equality test for the w-th letter in words X_i and Y_j (where $\{0\}_K$ denotes equality and $\{1\}_K$ denotes inequality). The server sums the entries in this matrix to compute the encrypted Hamming distance $d_i^j = \Delta(X_i, Y_j)$ between the words X_i and Y_j. Subsequently, the server sends Y_j blinded by a random value r multiplied by $d_i^j - \ell$, for all $0 \leq \ell \leq T - t$. If $0 \leq d_i^j \leq T - t$, then for some ℓ the value Y_j is not blinded at all. This allows the client to recover Y_j. Otherwise Y_j is blinded by some random value for every ℓ, and the client learns nothing.

3) Correctness and Security of the protocol from Figure 6: Assuming that in the subroutine `equality-matrix` the matrix f has been securely obtained, protocol 6 calculates a correct output. This can be concluded from the following facts: if $X_i \approx_t Y_j$ then (in step 3a) $\Delta(X_i, Y_j) \in \{\ 0\ \ldots\ T - t\ \}$, and therefore $\{0^k || Y_j\}_K$ is sent to the client. Privacy of the server is protected because in step 3a if $X_i \not\approx_t Y_j$ then $\Delta(X_i, Y_j) \notin \{0, \ldots T - t\}$ and therefore all values received by the client look random to him. Correctness and security proofs of this protocol resemble the proofs of the protocol presented in Figure 4 and are omitted here.

A. Implementing Subroutine `equality-matrix`

The first method to implement the subroutine `equality-matrix` is as follows. The client sends the letters of all his words to the server as encrypted vectors d_i^w: $\{0, \ldots |D| - 1\}$ (where $i \in \{1, \ldots n_C\}$ and $w \in \{1, \ldots T\}$) such that $d_i^w(v) = \{1\}_K$ if $v = x_i^w$, and $d_i^w(v) = \{0\}_K$ otherwise. This process can be described as sending encryptions of unary encoding of the letters of all his words. Subsequently the server defines the matrix as $f(w,i,j) = d_i^w(y_j^w)$. The main drawback of this method is that its bit complexity includes a factor $O(|D| \cdot n \cdot T + n^2 \cdot (T - t))$. However, the protocol is simple, and for small domains D (e.g., ASCII letters) it is efficient. For constant size D and $T \approx t$ the bit complexity of the

1) The client generates vectors d_i^w: $[0, \dots |D|-1]$ (where $i \in \{1, \dots n_C\}$ and $w \in \{1, \dots T\}$) such that: $d_i^w(v) = 1$ if $v = x_i^w$, and $d_i^w(v) = 0$ otherwise.
2) The matrix f is defined in the following way (for all $i, j \in \{1, \dots n\}$ and $w \in \{1, \dots T\}$):
   a) The client picks a random bit $b_{i,j}^w$.
   b) The server and the client perform 1–out–of–$|D|$ oblivious transfer as follows. The client constructs $h_{i,j}^w$, which is a vector $[0, \dots |D|-1]$ as follows:
   $h_{i,j}^w = [d_i^w(0) \oplus b_{i,j}^w, d_i^w(1) \oplus b_{i,j}^w, \dots d_i^w(|D|-1) \oplus b_{i,j}^w]$.
   The server wants to obtain a value from the vector $h_{i,j}^w$ with an index y_j^w. For that they perform the oblivious transfer protocol (where the server has an index and the client an array). Subsequently, the server obtains the value $h = h_{i,j}^w(y_j^w)$.
   c) The client sends $\{b_{i,j}^w\}_K$ to the server.
   d) $f(w,i,j) = \begin{cases} \{b_{i,j}^w\}_K, & \text{for } h = 0 \\ \{1 - b_{i,j}^w\}_K, & \text{for } h = 1 \end{cases}$

Fig. 7. Subroutine `equality-matrix` based on oblivious transfer

protocol reduces to $\tilde{O}(n^2 + n \cdot T)$ (which is significantly better than the bit complexity of the protocol from [3] in this situation).

The second implementation of the subroutine is shown in Figure 7. This implementation uses 1–out–of–q oblivious transfer. An oblivious transfer is a 2-party protocol, where a client has a vector of q elements, and the server chooses any one of them in such a way that the server does not learn more than one, and the client remains oblivious to the value the server chooses. Such an oblivious transfer protocol is described in [6]. The fastest implementation of oblivious transfer works in time $\tilde{O}(1)$.

The second version of the subroutine `equality-matrix` uses such an oblivious transfer in the following way. Let d_i^w be the unary encoding of x_i^w as defined above (in the description of the first method of implementation). The client chooses a random bit $b_{i,j}^w$. Next he constructs a vector $h_{i,j}^w$ which contains all bits of d_i^w, each blinded by the random bit $b_{i,j}^w$. In other words $h_{i,j}^w[x] = d_i^w(x) \oplus b_{i,j}^w$. Using an oblivious transfer protocol, the server requests the y_j^w-th entry in this vector, and obtains $d_i^w(y_j^w) \oplus b_{i,j}^w$. By the obliviousness, the client does not learn y_j^w, and the server does not learn any other entry. Subsequently, the client sends the encryption $\{b_{i,j}^w\}_K$ to the server. Based on this the server constructs $f(w,i,j) = \{d_i^w(y_j^w)\}_K$ as explained in the protocol.

1) Corollary: These protocols are in general less efficient in bit complexity than the improved protocol based on secret sharing (see Section V-B, Figure 5). The first protocol is efficient for small domains, but significantly less efficient for large ones. In the second protocol there are $n^2 \cdot T$ oblivious transfer calls. Moreover, at this stage, we do not foresee a way to improve these protocols. However, the protocols are interesting because they do not use generic 2-party computations. Furthermore, the techniques being used contain novel elements especially in the subroutine `equality-matrix`, that presents a technique for obtaining the encryption of a single bit using only one oblivious transfer.

VII. Summary and Future Work

In this paper we have presented a few protocols solving the FPM problem. The most efficient one works in a linear bit complexity with respect to the size of the input data and the security parameter. This is a significant improvement over existing protocols. The improvement comes at an expense of a factor n increase in time complexity (but only at the client).

Currently, we are investigating how to speed up the time complexity of the client by using error correcting coding techniques.

References

[1] M. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology — EUROCRYPT 2004.*, 2004, pp. 1–19.

[2] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," Cryptology ePrint Archive, Report 2003/235, 2003, http://eprint.iacr.org/.

[3] P. Indyk and D. Woodruff, "Polylogarithmic private approximations and efficient matching," in *The third Theory of Cryptography conference 2006*, vol. 3876 of LNCS, 2006, pp. 245–264.

[4] F. Boudot, B. Schoenmakers, and J. Traoré, "A fair and efficient solution to the socialist millionaires' problem," *Discrete Applied Mathematics*, vol. 111, no. 1–2, pp. 23–36, 2001. [Online]. Available: citeseer.ist.psu.edu/boudot01fair.html

[5] R. Fagin, M. Naor, and P. Winkler, "Comparing information without leaking it," *Communications of the ACM*, vol. 39, no. 5, pp. 77–85, 1996. [Online]. Available: citeseer.ist.psu.edu/article/fagin96comparing.html

[6] M. Naor and B. Pinkas, "Oblivious transfer and polynomial evaluation," in *Thirty-First Annual ACM Symposium on the Theory of Computing*, May 1999, pp. 245–254.

[7] L. Kissner and D. Song, "Privacy-preserving set operations," in *Advances in Cryptology — CRYPTO 2005.*, 2005, pp. 68–80.

[8] J.-H. Hoepman, "Private handshakes," in *4th Eur. Symp. on Security and Privacy in Ad hoc and Sensor Networks*, 2007.

[9] D. Balfanz, G. Durfee, N. Shankar, D. Smetters, J. Staddon, and H.-C. Wong, "Secret handshakes from pairing-based key agreements," in *24th IEEE Symposium on Security and Privacy*, Oakland, CA, May 2003, p. 180.

[10] C. Castelluccia, S. Jarecki, and G. Tsudik, "Secret handshakes from ca-oblivious encryption," in *In Advances in Cryptology - ASIACRYPT 2004: 10th International Conference on the Theory and Application of Cryptology and Information Security*, vol. 3329, December 2004, pp. 293–307.

[11] K. Du and M. Atallah, "Protocols for secure remote database access with approximate matching," in *the First Workshop on Security and Privacy in E-Commerce, Nov. 2000.*, November 2000. [Online]. Available: citeseer.ist.psu.edu/du00protocols.html

[12] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright, "Secure multiparty computation of approximations," *Lecture Notes in Computer Science*, vol. 2076, pp. 927+, 2001.

[13] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen, "On private scalar product computation for privacy-preserving data mining," *Lecture Notes in Computer Science*, vol. 3506, pp. 104–120, 2004.

[14] O. Goldreich, *Secure multi-party computation*. Cambridge University Press, 2002.

[15] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or a completeness theorem for protocols with honest majority," in *STOC*. ACM, 1987, pp. 218–229.

[16] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *STOC*, 2002, pp. 494–503.

[17] M. Naor and K. Nissim, "Communication complexity and secure function evaluation," *CoRR*, vol. cs.CR/0109011, 2001.

[18] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT 1999.*, May 1999, pp. 223–238.

[19] A. Shamir, "How to share a secret," in *Communications of the ACM, vol. 22, n.11*, November 1979, pp. 612–613.

Appendix

a) Correctness and security of the protocol from Figure 4: In this protocol the client encrypts all of letters of all of his words (with a unique secret key for every word) and sends the results to the server. Then for every couple of words (X_i, Y_j), the participants run the subroutine `find-matching`. In the subroutine firstly the server encrypts Y_j with some random secret key sk_j of symmetric encryption scheme. Later it divides sk_j into T shares (with threshold t) and for every letter in Y_j calculates $v_w = \{((x_i^w - y_j^w) \cdot r + \overline{s}^w)\}_K$. 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 the secret key using any combination of t out of the T elements $\{D_{sk}(v_w) | 1 \leq w \leq T\}$. He recognizes the secret key by the 0^k prefix, and the fact that decrypted by that secret key value is similar with one of the words from his set. If he has less than t correct secret shares then he cannot recover the secret key, and the retrieved data looks random to him (this 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.

The client input data is secure because all of the data received by the server is encrypted (using the 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 and he can reconstruct the secret $0^k || sk_j$ (and therefore, it can decrypt Y_j). If there is no element in X to which Y_j is similar then the client receives n 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 key. 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.

b) Correctness of the protocol from Figure 5: The first important issue appears in step 3 of the polynomial phase. Here the server prepares n groups of $[T+1]$–out–of–$[2 \cdot T - t + 1]$ shares $[\overline{s_j}^1, \overline{s_j}^2, \ldots \overline{s_j}^{2 \cdot T - t + 1}]$. From the jth group he can recover sk_j, and therefore, Y_j. During the creation of these shares the server uses the rule:

$$\text{for } w \in \{1, \ldots T\}\text{: if } y_i^w = y_m^w \text{ then } \overline{s_i}^w = \overline{s_m}^w. \qquad (1)$$

This rule is necessary because the first T shares from each group are later encoded as polynomials.

This secret sharing is used here in the same role as the t–out–of–T one. However if the t–out–of–T scheme is used, then it is impossible to choose the proper value of secrets (e.g., two matching, but different, words from Y, would have the same secret because of Rule 1). Secret shares $[\overline{s_i}^{T+1}, \ldots \overline{s_i}^{[2 \cdot T - t + 1]}]$ are chosen arbitrarily only to enable proper values of the secrets. To choose arbitrary secrets even for equal words (Y could be a multiset) $(T - t + 1)$ new shares (the ones that are sent in plaintext) is exactly enough. The role of shares $[\overline{s_i}^1, \ldots \overline{s_i}^T]$ is like in classical secret sharing. Because the last $T - t + 1$ shares are known, the first T shares work like a t–out–of–T secret sharing scheme.

Subsequently, in step 4, the server creates T polynomials of degree n in such a way that evaluating a polynomial on a corresponding letter from some word from Y results in a corresponding secret share. Later he sends the encrypted polynomials to the client. The client evaluates the polynomials on his words and achieves $\{v_i^w\}_K$ (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 T values $q_i^w = v_i^w + \overline{\tau_i}^w$, where $[\overline{\tau_i}^1, \overline{\tau_i}^2, \ldots \overline{\tau_i}^T]$ are tickets – 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 the linear property of LSS, if v_i^w is a correct secret share, then $q_i^w = v_i^w + \overline{\tau_i}^w$ is also a correct secret share. The client is trying to recover a secret for every received group of potential shares. However, for a proper reconstruction, he also needs shares that have been sent to him in plaintext by the server. These shares are always correct, but he needs to combine shares from the polynomial and ticket phases. Moreover, he does not know which shares from the polynomial phase correspond to the shares from the ticket phase. As a result, the client has to check all of the combinations (n^2). If the client combines non-fitting shares then he cannot recover the proper secret key (and therefore the proper word).

Hence, for $i, j \in \{1, \ldots n\}$, the client checks if he can reconstruct the secret key from the following shares:
$[q_i^1,\ q_i^2,\ \ldots\ q_i^T,\ \overline{s_j}^{T+1} + \overline{\tau_i}^{T+1},\ \overline{s_j}^{T+2} + \overline{\tau_i}^{T+2},\ \ldots$
$\overline{s_j}^{2 \cdot T - t + 1} + \overline{\tau_i}^{2 \cdot T - t + 1}]$.
If enough corresponding secret shares are in the group q_i, then the secret that could be recovered from them is $0^k || sk_m$ (because the secret of τ shares is 0). Hence, in step 9 the client recovers all of the secret keys that he has corresponding shares of.