Privacy-Preserving Matching Protocols for Attributes and Strings

Pu Duan, Sanmin Liu, Weiqin Ma, Guofei Gu and Jyh-Charn Liu
Department of Computer Science and Engineering,
Texas A&M University,College Station,TX,USA
{ dp1979, sanmin, weiqinma, guofei, liu }@cse.tamu.edu

Abstract

In this technical report we present two new privacy-preserving matching protocols for singular attributes and strings, respectively. The first one is used for matching of common attributes without revealing unmatched ones to each other. The second protocol is used to discover the longest common sub-string of two input strings in a privacy-preserving manner. Compared with previous work, our solutions are efficient and suitable to implement for many different applications, e.g., discovery of common worm signatures, computation of similarity of IP payloads.

Key word: privacy-preserving attribute matching, longest common sub-string, elliptic curve cryptosystem

1. Introduction

In this technical report we present two privacy-preserving matching protocols. The first one, privacy-preserving attribute matching protocol (*PPAM*), is used for matching of singular attributes. If two users, say Alice and Bob, both have their own attributes, they can use the protocol to discover their common attributes without revealing the unmatched ones to each other. For example, Alice's attributes that are not in the intersection are not revealed to Bob. In addition, all transmitted messages are confidential, i.e., outside adversaries cannot know the attributes, the matching result, etc. In PPAM matched attributes (IP addresses, numbers, words, etc) have same values. Our second protocol, privacy-preserving longest common sub-string computation protocol (*PPLS*), is used to discover to the longest common sub-string of two input strings. This protocol is more complex since it cannot simply depends on test of equal values (i.e., input strings are usually different), and can be viewed as a fuzzy matching method. PPLS provides full confidentiality against outside adversaries (i.e., outsiders cannot know input strings) and partial confidentiality against insider adversaries (i.e., an insider may know more than the longest common sub-string after executing the protocol with a user). PPLS can be used in many different applications, e.g., discovery of common signatures for worm signature generation tools, computation of similarity of IP payloads. In the follows we present the detail of the two protocols.

2. Privacy-Preserving Matching Protocols

2.1 Privacy-Preserving Attribute Matching Protocol

Before executing PPAM, some public parameters, (Q, q, G_1, H_1), need to be generated and published. q is a large prime number, G_1 denotes an additive cyclic group of prime order q, Q is a base element of G_1, H_1 is a hash function that maps a string with arbitrary length to a string with fixed length. Here G_1 is selected in such a way that *Discrete Logarithm Problem (DLP)* [10] is assumed to be hard on it. In our design G_1 is chosen as a group of points on a specific elliptic curve [11]. Three functions are called in PPAM to achieve the matching: an encryption function E, a decryption function D and a modification function M. In the follows we describe the protocol in a scenario with two users, say Alice and Bob. To simplify the description, we only describe Alice's matching procedure since Bob's is similar. First Alice generates her pairwise matching keys (*ME_A*, *MD_A*). *ME_A* is used in E and the latter one is in D. Then Alice takes her input attributes *ATT(A)* and uses the encryption function E to encrypt them to get the ciphertexts $E_{ME\_A}(ATT(A))$. The ciphertexts are then sent to Bob. After receiving $E_{ME\_A}(ATT(A))$, Bob modifies Alice's ciphertexts based on the modification function M to get $MTT(AB) = M(ATT(B), E_{ME\_A}(ATT(A)))$, where *ATT(B)* denotes Bob's input attributes. Bob then sends the modified ciphertexts back to Alice. After receiving the modified encrypted attributes, Alice uses the decryption function D and *MD_A* to decrypt the

modified ciphertexts as $D_{MD\_A}(MTT(AB))$. Common attributes between $ATT(A)$ and $ATT(B)$ will be discovered after the decryption.

Suppose $ATT(A) = \{a_0, a_1, \ldots, a_n\}$ and $ATT(B) = \{b_0, b_1, \ldots, b_m\}$. n and m are two integers. Alice and Bob generate their own matching keys based on the base element $Q \in G_1$. Alice's pairwise matching keys are generated as $ME\_A = MD\_A \times Q$, where $MD\_A \in Z_q$ and $ME\_A \in G_1$. *Bob* also generates his key pair in a similar way, i.e., $ME\_B = MD\_B \times Q$. Before exchange their attributes for matching, Alice generates two large random integers r_A, $r_A' \in Z_q$. Bob also generates his random integers r_B, $r_B' \in Z_q$. To encryption her attributes, Alice needs to transfer her attributes to large integers first, i.e., $H_1(ATT(A)) = [H_1(a_0), H_1(a_1) \ldots H_1(a_n)]$. Then she uses the encryption key ME_A to generate her ciphertexts by executing encryption function E: $E_{ME\_A}(ATT(A)) = [CA_0, CA_1, \ldots, CA_n]$, where $\forall\, i \in [0, n]$, $CA_i = (CA_{if}, CA_{ib}) = (r_A Q, r_A \times H_1(a_i) \times ME\_A)$. *Bob* generates his ciphertexts in a similar way: $E_{ME\_B}(ATT(B)) = [CB_0, CB_1, \ldots, CB_m]$, where $\forall\, j \in [0, m]$, $CB_j = (CB_{jf}, CB_{jb}) = (r_B Q, r_B \times H_1(b_j) \times ME\_B)$. The detail of the matching protocol is as follows.

Privacy-Preserving Attributes Matching Protocol (PPAM)

Input: Alice's attribute set ATT(A), Bob's attribute set ATT(B)

Output: the intersection between ATT(A) and ATT(B)

1. *Alice: generate* $E_{ME\_A}(ATT(A))$
2. *Bob: generate* $E_{ME\_B}(ATT(B))$
3. *Alice* $\rightarrow$ *Bob:* $E_{ME\_A}(ATT(A))$
4. *Bob* $\rightarrow$ *Alice:* $E_{ME\_B}(ATT(B))$
5. *Alice: execute modification function M on ATT(A) and* $E_{ME\_B}(ATT(B))$ *to get* $MTT(BA) = M(ATT(A), E_{ME\_B}(ATT(B))) = [(CB_{0f}', CB_{1f}', \ldots, CB_{nf}'), (CB_{0b}', CB_{1b}', \ldots, CB_{mb}')]$, *for each* $i \in [0, n]$, $j \in [0, m]$, $CB_{ij}', CB_{jb}' = [r_A' \times H_1(a_i) \times CB_{jf}], [r_A' \times CB_{jb}]$
6. *Bob: execute modification function M on ATT(B) and* $E_{ME\_A}(ATT(A))$ *to get* $MTT(AB) = M(ATT(B), E_{ME\_A}(ATT(A))) = [(CA_{0f}', CA_{1f}', \ldots, CA_{mf}'), (CA_{0b}', CA_{1b}', \ldots, CA_{nb}')]$, *for each* $i = 0$ *to* $i = n$, $j = 0$ *to* $j = m$, $CA_{jf}', CA_{ib}' = [r_B' \times H_1(b_j) \times CA_{if}], [r_B' \times CA_{ib}]$
7. *Alice* $\rightarrow$ *Bob: MTT(BA)*
8. *Bob* $\rightarrow$ *Alice: MTT(AB)*
9. *Alice: execute decryption function D on MTT(AB) to get* $D_{MD\_A}(MTT(AB))$, *where for* $i = 0$ *to* $i = n$, $j = 0$ *to* $j = m$, *test whether* $MD\_A \times CA_{jf}' = CA_{ib}'$. *If yes, put the corresponding attribute* $H_1(a_i)$ *into* SI_A *as the intersection set derived by Alice*
10. *Bob: execute decryption function D on MTT(BA) to get* $D_{MD\_B}(MTT(BA))$, *where for* $i = 0$ *to* $i = n$, $j = 0$ *to* $j = m$, *test whether* $MD\_B \times CB_{if}' = CB_{jb}'$. *If yes, put the corresponding attribute* $H_1(b_j)$ *into* SI_B *as the intersection set derived by Bob*

2.2 Privacy-Preserving Longest Common Sub-String Computation Protocol

PPLS is to compute the longest common sub-string of two input strings. The main idea is to use PPAM as the building block and discover the longest common sub-string in a privacy-preserving manner. First we need extract attributes as sub-strings with a minimum meaningful length from the original string, e.g., 5-gram sub-strings. It is because that matched n-grams with length shorter than the minimum meaningful length, e.g., matched 2-grams or even 1-grams, are not very useful. These initial attributes are matched with the same-size attributes extracted from others at the first phase. If common attributes are found, the protocol further computes all possible longer sub-strings (based on the matched sub-strings and the original string) and does a second-round matching. It is not trivial to design the second phase of the protocol because how to reveal the position information of each matched attribute without compromising the user's privacy is a hard problem. There are two simple approaches for this purpose: (1) one user's position information of all matched attributes is directly revealed to his partner at the first phase so that his partner can easily reconstruct all the matched sub-strings to find the longest one. Though this approach is efficient since only one round of matching is needed, it is not secure enough because the reconstructed sub-strings may already unveil the user's personal information. For example, one user's several unlinked shorter sub-strings may reveal a longer sub-string of another user. (2) the other approach is to match all possible sub-strings from the longest to the shortest. i.e., directly feed sub-strings as inputs for matching. Users may execute the matching process for several rounds (decrease the length of the input sub-strings when no

matched one is found at the current length) until a common sub-string is found. This approach is more secure since no extra information is revealed except of the longest common sub-string, but it is not efficient enough since more rounds of matching are needed. The goal of our design is to find the balance between the privacy protection and the efficiency. First we describe several observations for the design of PPLS as follows.

Observations:

(1) If a user's input string has repetitive parts, e.g., …abcde…abcde…, then only one part needs to be extracted for matching.
(2) After common attributes are discovered (the first phase), though a user does not know the position information of the matched attributes of his partner, he can list all possible common sub-strings based on his own inputs.
(3) There are two possible situations in which a user's privacy may be compromised, i.e., one user's several unlinked shorter sub-strings reveal a longer sub-string of another user (the shorter sub-string literally compose the longer sub-string) or vice versa. To mitigate this problem, a user may input numerous random points for matching such that the protected sub-strings cannot be exhaustively matched by other unlinked shorter sub-strings or a longer sub-string.

Our main idea is to treat the attributes that comprise a possible sub-string as a sub-set, and use PPAM to discover common attributes of every two sub-sets. For example, Alice has computed sub-sets A_1, A_2 and Bob has his sub-sets B_1, B_2. Alice uses PPAM to discover the common intersection as $SI(A_1, B_1)$, $SI(A_1, B_2)$, $SI(A_2, B_1)$ and $SI(A_2, B_2)$; while Bob also does his own matching. During the process, a user may use a randomization function to add random points into the ciphertexts when encryption function E and modification function M are executed. The detail of the randomization function is introduced below. A trick here is that when a user receives the modified ciphertexts from his partner and wants to do the decryption (Step 9, 10 in PPAM), he should test the equality between every two points following the sequence of the attributes that comprise the corresponding sub-string. For example, when Alice wants to find $SI(A_1, B_1)$ and $SI(A_1, B_2)$, she should choose the first point whose corresponding attribute is the first attribute in the sub-string of A_1, then the second point and so on. If the first attribute has a matched one in B_1 but the second has a matched attribute in B_2, there is no need to record these two attributes; otherwise, record the matched and sequenced attributes. In the follows we first present the detail of the randomization function.

Randomization Function: R

Input: a random integer s
Output: s random points on G_1, denoted by $P_0, P_1, \ldots, P_{s-1}$

During the execution of encryption function E and modification function M, a user may generate numerous random points and put them into the ciphertexts and modified ciphertexts to obtain redundancy. For example, in PPAM when Alice encrypts $ATT(A)$ to get $E_{ME\_A}(ATT(A)) = [CA_0, CA_1, \ldots, CA_n]$, she could call the randomization function to generate extra s points and attach them with the real ciphertexts, i.e., $E_{ME\_A}(ATT(A))' = [CA_0, CA_1, \ldots, CA_n, CA_0', \ldots, CA_s']$, where $\forall i \in [0, s]$, $CA_i' = (CA_{if}', CA_{ib}') = (r_A Q, P_i)$, P_i is generated from R. After adding random points into the ciphertexts, a user's partner cannot tell whether a point is a ciphertext generated from a real attributes or a faked random point. Now we present the detail of our second matching protocol.

Privacy-Preserving Longest Common Sub-String Computation Protocol(PPLS)

Input: Alice's string S_A, Bob's string S_B, minimum meaningful length l, and all other security parameters presented in PPAM
Output: the longest common sub-string of S_A and S_B

Phase 1

1. *Alice: extract attributes with minimum length l from S_A, the set of the attributes is denoted by T(A)*
2. *Bob: extract attributes with minimum length l from S_B, the set of the attributes is denoted by T(B)*
3. *Alice $\leftrightarrow$ Bob: Alice and Bob implement PPAM and discover the common attributes in the intersection of T(A) and T(B), denoted by T(C)*

Phase 2

4. *Alice: analyze S_A and T(C), take the following steps: (a) for any matched attributes a_i, $a_j \in T(C)$, combine them and generate a longer sub-string $sub_{ij} = a_i a_j$ if a_i and a_j have adjacent positions in S_A, the set of the computed sub-strings is denoted by SUB(A), (b) for any sub-strings sub_i, $sub_j \in SUB(A)$,*

if sub_i is still a sub-string of sub_j, only remain sub_j (e.g., if sub_i = abcdef, sub_j = abcdefg, remain sub_j), (c) suppose all the remained sub-strings comprises the set SUB_R_(A), for each sub-string $sub_k \in$ SUB_R_(A), use all the matched attributes that comprise sub_k to build an attribute sub-set S_SET(sub_k), all attributes in S_SET(sub_k) are sequenced randomly and all the sub-sets are denoted by S_SET(A), (d) Alice uses PPAM to encrypt all the sub-sets S_SET(A) and calls he randomization function R to add random points into each ciphertext sub-set, the final ciphertext sub-sets are denoted by C_SET(A), (d)Alice inserts an infinity point chosen from the elliptic curve point group G_1 as a symbol to separate every two different ciphertext sub-sets in C_SET(A) and sends all the computed ciphertext sub-sets with the infinity points to Bob

5. *Bob: compute his ciphertext sub-sets with similar procedures and send his to Alice as C_SET(B)*
6. *Alice: modify Bob's ciphertexts sub-sets based on her own attribute sub-sets M(C_SET(B), S_SET(A)), call randomization function R to add redundant points into the group generated by her own attributes, input an infinity point to separate every two ciphertext sub-sets generated by her own attributes, send the modified ciphertexts sub-sets back to Bob*
7. *Bob: modify Alice's ciphertexts sub-sets based on his own attribute sub-sets M(C_SET(A), S_SET(B)) with the similar procedures and send the modified ciphertext sub-sets back to Alice*
8. *Alice: decrypt the modified ciphertext sub-sets, discover all matched attributes that satisfy two conditions: (a) they are in a single sub-set on both of the two users (b) they have adjacent position in Alice's sub-strings,*
9. *Bob: discover matched attributes with similar procedures*

Phase 3

10. *Alice: combine all continuous matched attributes, build all possible common sub-strings, treat each sub-string as an attribute and execute PPAM with Bob to find the matched sub-string with the longest length*
11. *Bob: combine all continuous matched attributes, build all possible common sub-strings, treat each sub-string as an attribute and execute PPAM with Alice to find the matched sub-string with the longest length*

Since PPLS is built based on PPAM, outsiders cannot compromise a user's privacy by eavesdropping transmitted messages, i.e., String Confidentiality is guaranteed. But for malicious insiders, PPLS only holds partial confidentiality, named as Partial Sub-String Confidentiality, which will be analyzed in detail in the next section. PPLS can also be used to find common sub-sequences (*CSS*) of input strings. In this case this protocol also holds full confidentiality, String Confidentiality, for outside adversaries, and Partial Sub-String Confidentiality for inside adversaries.

2.3 Security Analysis

In our design we assume that protocols users are collaborators and behave correctly. No malicious users actively forge attributes, intercept traffic, etc, i.e., no *active attacks* are considered. On the other hand, we consider *passive attacks* that aim at compromising confidentiality of sensitive information by analyzing transmitted messages, e.g., dictionary attack. We also consider both outside adversaries and inside adversaries as follows.

Outside adversary: a malicious user that is an outsider of a correlation process. An outside adversary does not participate in the matching process and knows nothing about the correlation.

Inside adversary: a malicious user that participates in a matching process. The adversarial insider may have some matched attributes with the targeted victim and try to discover other unmatched attributes the victim has.

Based on the adversary model and attack model introduced above, we claim that our PPAM provides the following security properties: Private Set Intersection and Attribute Confidentiality. The detailed proof is presented in Appendix.

(1) Private Set Intersection: matching peers only discover matched attributes in the intersection set of their input attributes. This property is guaranteed by PPAM. Say Alice has attribute set A and Bob has set B. They did matching process with each other and found their intersection I_{AB}. If attribute $a_i \in A$ and $a_i \notin I_{AB}$, Bob cannot learn a_i by launching any passive attacks and vice versa.

(2) Attribute Confidentiality: an adversary cannot learn what attribute a peer owns if this adversary does not own the same attributes. Based on the hardness of ECDLP, any polynomial-time adversary, either outside adversary or inside adversary, cannot compromise attribute confidentiality without compromising attribute owners. Say Alice has attribute set A. If $a_i \in A$ and Bob does not have a_i, Bob cannot learn a_i by launching any passive attacks.

Based on the adversary model and attack model introduced above, we claim that our PPLS holds the following properties: Sub-String Confidentiality for outside adversaries and Partial Sub-String Confidentiality for inside adversaries. Here we also assume that adversaries do not launch active attack, e.g., forge sub-strings.

(3) String Confidentiality: an outside adversary cannot learn what input string a peer owns or what results (the longest common sub-string) two peers obtain. Based on the hardness of ECDLP, any polynomial-time outside adversary, cannot compromise sub-string confidentiality without compromising sub-string owners. The detail of the proof of this property is similar as that of Attribute Confidentiality and will be omitted here.

(4) Partial Sub-String Confidentiality: an inside adversary cannot learn what sub-string a peer owns if the insider does not have the corresponding attributes, but he may know what sub-string a peer possibly has if the adversary owns the corresponding attributes. That is said, if the matched attributes comprise same sub-strings on both the user's and adversary's side, the adversary will learn this fact; otherwise, the adversary will only know that the user possibly has the sub-string (determined by the randomization function R).

3. Security Proof

Attribute Confidentiality: attribute confidentiality means that an adversary cannot learn what attributes a user owns. Based on the hardness of ECDLP, in our protocol any polynomial-time adversary cannot compromise attribute confidentiality without compromising other valid owners of the targeted attributes. Suppose there is an adversary A who aims at finding out the contents of some targeted attribute a. A may communicate with legitimate owners of the targeted attribute a in network G, corrupt some valid users and obtain their attributes. Here we use U^T to denote the set of users who own the targeted attribute a. A picks a target user $u^T \in U^T$, and wants to find out the attribute by communicating with u^T. Now we define the *Attribute Detection Game* for a randomized, polynomial-time adversary A as follows:
Step 1: The adversary A communicates with owners of the targeted attribute based on its own choice. A may compromise certain user $U^C \subseteq U$ and obtain their attributes, where U^C denotes the set of compromised users and U denotes the whole user set, where $U^C \cap U^T = \varnothing$

Step 2: A selects a target user $u^T \in U^T$, where users in U^T own the targeted attribute a.

Step 3: A generates a' by communicating with u^T.

We say that A wins the Attribute Detection Game when $a' = a$. Now we define the following probabilities:

$G_A = Pr[A \text{ wins Attribute Detection Game}]$

When A does not compromise any valid owner of the targeted attribute a, the above probability becomes:

$G'_{A|(U^C \cap U^T)=\varnothing} = Pr[A \text{ wins Attribute Detection Game} \mid (U^C \cap U^T) = \varnothing]$

Now we can define *Attribute Confidentiality* of our protocol. Suppose A never compromises a valid owner of the target a. In other words, $(U^C \cap U^T) = \varnothing$. Our protocol is said to have Attribute Confidentiality if $G'_{A|(U^C \cap U^T)=\varnothing}$ is negligible for any adversary A. To prove that PPAM has the property, we first present the following theorem that has been proved in [1]:

Theorem 1(Lemma 2 and Lemma 3 [1]):
C's Privacy is preserved and S's Privacy is preserved.

Now we compare our protocol with the protocol presented in [1] and prove the following corollary:

Corollary 2

If private set matching model proposed in [1] has preserved user's privacy, then PPAM holds Attribute Confidentiality.
Proof: the difference between PPAM and the private set matching protocol in [1] is that PPAM generates elliptic curve point as the user's input ciphertexts and uses $E_{ME\_A}(ATT\_(A))$ and $E_{ME\_B}(ATT(B))$ as user's ciphertexts instead of the equation ciphertexts in [1]. Because all ciphertexts are randomized with single-use random number r_A/r_B and r_A'/r_B' same as the randomization in [1], the ciphertexts hold the same security property as the ciphertexts hold in [1] based on the assumption that DLP is hard in G_1 (ECDLP). Thus PPAM holds Attribute Confidentiality if the matching scheme [1] preserves user's attribute privacy.

Private Set Intersection
Two users *A* and *B* in a matching process only know the elements in the intersection of their attribute sets, i.e. one user's attributes that are not in the intersection are not revealed to the other. The proof is still based on the ECDLP and is omitted here.

String Confidentiality
Two users *A* and *B* are in a matching process, and an outside adversary *T* cannot know what strings A and B have and what the longest common sub-string A and B obtain, without compromising A or B. The proof is still based on the ECDLP and is omitted here.

Partial Sub-String Confidentiality
Partial Sub-String Confidentiality means that an inside adversary cannot learn earn what sub-string a peer owns if the insider does not have the corresponding attributes, but he may know what sub-string a peer possibly has if the adversary owns the corresponding attributes. The proof of the first part is similar to the proof of Attribute Confidentiality and is omitted here. For the second part, it is said, (1) if the matched attributes comprise same sub-strings on both the user's and adversary's side, the adversary will learn this fact; (2) otherwise, the adversary will only know that the user possibly has the sub-string (determined by the randomization function *R*). We now prove (2). We define the *Sub-String Detection Game* for a randomized, polynomial-time inside adversary *A* as follows:
Step 1: The adversary *A* communicates with owners of the targeted sub-string based on its own choice. *A* may compromise certain user $U^C \subseteq U$ and obtain their sub-strings, where U^C denotes the set of compromised users and *U* denotes the whole user set, where $U^C \cap U^T = \varnothing$

Step 2: *A* selects a target user $u^T \in U^T$, where users in U^T own the targeted sub-string *s*.

Step 3: *A* generates *s'* by communicating with u^T.

We say that *A* wins the Sub-String Detection Game when *s'* = *s*. Now we define the following probabilities:

$G_A = Pr[A \text{ wins Sub-String Detection Game}]$

When *A* does not compromise any valid owner of the targeted sub-string *s*, the above probability becomes:

$G'_{A|(U^C \cap U^T)=\varnothing} = Pr[A \text{ wins Sub-String Detection Game} \mid (U^C \cap U^T) = \varnothing]$

When $s \notin S(A)$, *S(A)* denotes *A*'s whole sub-string set, the above probability becomes:

$G'_{A|(U^C \cap U^T)=\varnothing \,\&\, s \notin S(A)} = Pr[A \text{ wins Sub-String Detection Game} \mid (U^C \cap U^T) = \varnothing \,\&\, s \notin S(A)]$

Now we can define *Partial Sub-String Confidentiality* of our protocol. Suppose *A* never compromises a valid owner of the target *s* and the matched attributes do not comprise *s*. In other words, $(U^C \cap U^T) = \varnothing$ and $s \notin S(A)$, which denotes *A*'s whole sub-string set. Our protocol is said to have Partial Sub-String Confidentiality if $G'_{A|(U^C \cap U^T)=\varnothing \,\&\, s \notin S(A)}$ is negligible for any adversary *A*. To prove that PPLS has the property, we present the following theorem

Theorem 2
If private set matching model proposed in [1] has preserved user's privacy, then PPLS holds Partial Sub-String Confidentiality.
Proof: the difference between PPLS and the private set matching protocol in [1] is that PPLS generates elliptic curve point as the user's input ciphertexts and uses $E_{ME\_A}(ATT\_(A))$ and $E_{ME\_B}(ATT(B))$ as user's ciphertexts instead of the equation ciphertexts in [1]. Because all ciphertexts are randomized with single-use random number r_A/r_B and r_A'/r_B' same as the randomization in [1], the ciphertexts hold the same

security property as the ciphertexts hold in [1] based on the assumption that DLP is hard in G_1 (ECDLP). In addition, since $s \notin S(A)$ and all the attributes of s has been randomized by function R, i.e., redundant points are added into the $S\_SET(s)$, the adversary only has a probability to win this game by guessing s from n points (suppose n is the number of points generated by R). Thus PPLS holds Partial Sub-String Confidentiality if n is large enough.

4. Related Work

Freedman *et al*. [1] proposed the first two-party private set matching protocol. In their scheme a user first transfers matching attributes to roots of a chosen equation and encrypts them with a homomorphic encryption scheme. Then encrypted roots are sent to the other user, where he encrypts his own roots (based on his own attributes) and combine the two ciphertexts together based on same homomorphc encryption scheme. After modified ciphertexts are returned to the first user, he can discover the common attributes by decrypting them (i.e., the equation is computed to 0 with common attributes as common roots). Compared with their solution, our protocol also depends on a homomorphic encryption scheme. The main difference is that our solution does not need to produce an equation with correponsing roots and is easier to implement (i.e., no need to map equation roots to homormorphic ciphertexts). Shin *et al*. [2] proposed a privacy-enhanced matchmaking protocol which supports forward privacy of user's identities, and matching wishes, etc. Their protocol was based on the password-authenticated key exchange protocols [3]. Computing cost is an important concern. Kissner and Song [4] proposed a solution based on polynomial representations to construct privacy-preserving set operations (union, intersection, cardinality of intersection, and multiplicity test). A threshold homomorphic cryptosystem based protocol proposed in [5] for the set intersection and set matching further reduced the computing complexity. The proposed protocol in [5] was based on the combination of secret sharing and homomorphic encryption to reduce the computing cost of the cardinality of set intersection. The privacy-preserving set intersection protocol proposed in [6] was based on bilinear maps. Most of the protocols mentioned above are only designed to discover common singular attribute and cannot compute common parts of special attributes, e.g., strings. Another type of technique that can be used to achieve privacy-preserving attribute matching is secure multi-party computation (*SMC*). The idea was first proposed in [7]. The idea of secure computation is to enable a set of untrusting parties to compute certain function based on their own private inputs without revealing their private information except of the common result of the function. The Fairplay two-party computation system proposed in [8] implemented generic secure function evaluation and to solve several secure computation problems, e.g., Millionaires problem. The FairplayMP proposed in [9] was extended from Fairplay to achieve secure multi-party computation. Compared with our protocols, SMC solutions are usually not practical because of their high computing costs, and rigorous security requirements of user data control.

Reference

[1] M. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection", In *Advances in Cryptology-EUROCRYPT'04*, volume 3027 of LNCS, pages 1-19, 2004.

[2] J. S. Shin and V. D. Gilgor, "A New Privacy-Enhanced Matchmaking Protocol", *NDSS Symposium 2008*.

[3] J. Katz, R. Ostrovsky, and M. Yung, "Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords", *Eurocrypt 2001*.

[4] L. Kissner and D. Song, "Privacy-preserving set operations", In *Advances in Cryptology – CRYPTO'05*, volume 3621 of *LNCS*, pages 241–257, 2005.

[5] Yingpeng Sang, Hong Shen, Yasuo Tan, and Naixue Xiong, "Efficient protocols for privacy preserving matching against distributed datasets", In *8th International Conference of Information and Communications Security*, volume 4307, pages 210 – 227, December 2006.

[6] Yingpeng Sang, Hong Shen, "Privacy Preserving Set Intersection Based on Bilinear Groups", *ACSC*, 2008.

[7] A. C. Yao, "Protocols for secure computations", In P*roceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, 1982.

[8] D. Malkhi, N. Nisan, B. Pinkas and Y. Sella, "Fairplay – A Secure Two-Party Computation System", *13th USENIX Security Symposium*, pages 287-302, 2004.

[9] A. B. David, N. Nisan and B. Pinkas, "Fairplay – A System for Secure Multi-Party Computation", *CCS'08*, 2008.

[10] Lawrence C. Washington, "Elliptic Curves: Number Theory and Cryptography," published by Chapman & Hall/CRC, 2003.

[11] Shamus Software Ltd. "MIRACL Library," URL: http://indigo.ie/~mscott/