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].

PPAM consists of three major functions: an *encryption function* $\boldsymbol{E}$, a *decryption function* $\boldsymbol{D}$ and a *modification function* $\boldsymbol{M}$. Let $A = (a_0, a_1, \ldots, a_n)$ and $B = (b_0, b_1, \ldots, b_m)$ respectively represent the set of attributes owned by Alice and Bob. The syntax and formats of these attributes have been unified based on their agreement prior to matching. First Alice and Bob need to generate their own secret parameters. Alice computes an elliptic curve point Q_A on E for encryption purposes using a secret number k_A and the base point P: $Q_A = k_A P$ (i.e., the standard *point multiplication* in ECC), where $k_A \in Z_q$ and $Q_A \in G_1$. Due to the well known ECDLP, k_A cannot be deduced, given Q_A and P. To guarantee that adversaries cannot find any attribute by trying every possibility (the information space of alert attributes is usually not very large) in a dictionary attack, Alice also needs to generate three large random integers r_{A1}, r_{A2}, $r_{A3} \in Z_q$ for information hiding (i.e., mapping attribute values to elements on a larger group). For example, an attribute a_i cannot be

deduced from $(r_{A1}a_i + r_{A2})P$ by any passive attack because r_{A1} and r_{A2} are large enough to guarantee the ECDLP. Following a similar argument, Bob also needs to generate k_B, $Q_B = k_BP$, and r_{B1}, r_{B2}, $r_{B3} \in Z_q$.

Now we introduce the basic procedures of the matching protocol from Alice's side. First Alice uses $\boldsymbol{E}$ to generate the ciphertexts $\boldsymbol{E}(A) = [E(a_0), E(a_1), \ldots, E(a_n)]$ on the (hashed) input attributes A, where each element $E(a_i)$ consists of two elliptic curve points as its "front" point and "back" point. The "front" points are produced by associating Alice's chosen random numbers with base point P. The "back" points are produced by associating both the random numbers and Alice's attributes with her computed point Q_A, where Alice's attributes cannot be deduced because of ECDLP. Alice then sends $\boldsymbol{E}(A)$ to Bob through the PAC communicator. After receiving $\boldsymbol{E}(A)$ Bob uses $\boldsymbol{M}$ and his own attributes B to produce $\boldsymbol{M}[B, \boldsymbol{E}(A)]$, i.e., *modification* of $\boldsymbol{E}(A)$. The modification function $\boldsymbol{M}$ allows Bob to (1) associate both his own attributes B and one chosen random number with the "front" points of $\boldsymbol{E}(A)$, and (2) associate the same random number with the "back" points of $\boldsymbol{E}(A)$. Again, Bob's attributes cannot be computed from the modified "back" points because of ECDLP. Then Bob sends his $\boldsymbol{M}[B, \boldsymbol{E}(A)]$ back to Alice, who can use $\boldsymbol{D}$ to decrypt-and-then-evaluate the "front" and "back" points. After the decryption (i.e., eliminating the difference between the "front" points and "back" points caused by Alice's own secret number and random number) if a pair of "front" point and "back" point are equal, Alice and Bob have an identical/matched attribute. Otherwise, their attributes are different. Bob has symmetric operations as Alice and therefore will not be further repeated here. Details of PPAM are discussed as follows.

Protocol I: Privacy-Preserving Attribute Matching

Input: A, B

Output: the intersection between A and B

1. *Alice: generate* $\boldsymbol{E}(A) = [E(a_0), E(a_1), \ldots, E(a_n)]$, $\forall i \in [0, n]$, $E(a_i) = [E(a_i)_f, E(a_i)_b]$, where $E(a_i)_f = r_{A1}\, r_{A2}^{-1} P$, and $E(a_i)_b = [r_{A1}H_1(a_i) + r_{A2}]Q_A$

   Bob: generate $\boldsymbol{E}(B) == [E(b_0), E(b_1), \ldots, E(b_m)]$, $\forall j \in [0, m]$, $E(b_j) = [E(b_j)_f, E(b_j)_b]$, where $E(b_j)_f = r_{B1}\, r_{B2}^{-1} P$, and $E(b_j)_b = [r_{B1}H_1(b_j) + r_{B2}]Q_B$
2. *Alice* $\leftrightarrow$ *Bob: Alice sends E(A) to Bob and Bob sends E(B) to Alice*
3. *Alice: compute* $\boldsymbol{M}[A, \boldsymbol{E}(B)]$

   $\boldsymbol{M}[A, \boldsymbol{E}(B)] = [M(a_0)_f, M(a_1)_f, \ldots, M(a_n)_f], [M(b_0)_b, M(b_1)_b, \ldots, M(b_m)_b]$

   $\forall i \in [0, n]$, $M(a_i)_f = H_1[r_{A3}H_1(a_i)(r_{B1}r_{B2}^{-1})P + r_{A3}P] = H_1[(r_{A3}r_{B1}r_{B2}^{-1}H_1(a_i) + r_{A3})P]$

   $\forall j \in [0, m]$, $M(b_j)_b = r_{A3}[(r_{B1}H_1(b_j) + r_{B2})Q_B] = [r_{A3}r_{B1}H_1(b_j) + r_{A3}r_{B2}]Q_B$

   Bob: compute $\boldsymbol{M}[B, \boldsymbol{E}(A)]$

   $\boldsymbol{M}[B, \boldsymbol{E}(A)] = [M(b_0)_f, M(b_1)_f, \ldots, M(b_m)_f], [M(a_0)_b, M(a_1)_b, \ldots, M(a_n)_b]$

   $\forall j \in [0, m]$, $M(b_j)_f = H_1[r_{B3}H_1(b_j)(r_{A1}r_{A2}^{-1})P + r_{B3}P] = H_1[(r_{B3}r_{A1}r_{A2}^{-1}H_1(b_j) + r_{B3})P]$

   $\forall i \in [0, n]$, $M(a_i)_b = r_{B3}[(r_{A1}H_1(a_i) + r_{A2})Q_A] = [r_{B3}r_{A1}H_1(a_i) + r_{B3}r_{A2}]Q_A$
4. *Alice* $\leftrightarrow$ *Bob: Alice sends* $\boldsymbol{M}[A, \boldsymbol{E}(B)]$ *to Bob, and Bob sends* $\boldsymbol{M}[B, \boldsymbol{E}(A)]$ *to Alice*
5. *Alice: compute* $\boldsymbol{D}\{\boldsymbol{M}[B, \boldsymbol{E}(A)]\}$*: for* $i = 0$ *to* n*, compute* $H_1[k_A^{-1}r_{A2}^{-1}M(a_i)_b]$ /*the decryption function */

   for $i = 0, 1, \ldots n$, $j = 0, 1, \ldots m$*, test whether* $H_1[k_A^{-1}r_{A2}^{-1}M(a_i)_b] = M(b_j)_f$*, if yes,* a_i *is a matched attribute in the intersection*

   Bob: compute $\boldsymbol{D}\{\boldsymbol{M}[A, \boldsymbol{E}(B)]\}$*: for* $j = 0$ *to* m*, compute* $H_1[k_B^{-1}r_{B2}^{-1}M(b_j)_b]$

   for $i = 0, 1, \ldots n$, $j = 0, 1, \ldots m$*, test whether* $H_1[k_B^{-1}r_{B2}^{-1}M(b_j)_b] = M(a_i)_f$*. if yes,* b_j *is a matched attribute in the intersection*

We use the multiplications of large random numbers with ECC points to protect user's attribute attributes from dictionary attacks, i.e., $H_1(a_i)$ cannot be deduced from $E(a_i)_b = [r_{A1}H_1(a_i) + r_{A2}]Q_A$. In addition, we also use H_1 on $\boldsymbol{M}[A, \boldsymbol{E}(B)]$ and $\boldsymbol{M}[B, \boldsymbol{E}(A)]$ such that Alice (or Bob) cannot further manipulate the received ciphtertexts as ECC points. Otherwise, inside adversaries, e.g., Alice, could launch dictionary attack to try every possible values for attributes to get an equivalent equation between $H_1[k_A^{-1}r_{A2}^{-1}M(a_i)_b]$ and $M(b_j)_f$.

If n = m, both Alice and Bob need (4n + 6) (i.e., O(n)) point multiplications to compute the intersection: one multiplication to associate secret number with the base point, (n + 2) multiplications to generate the ciphertexts (step 1 or step 2), (2n + 2) multiplications to execute M on received ciphertexts (step 5 or step

6), (n + 1) multiplications to execute D on the modified ciphertexts (step 9 or step 10) for decryption. 2.2 Privacy-Preserving Longest Common Sub-String Computation Protocol

2.2 Privacy-Preserving Longest Common Sub-String Computation Protocol

PPLS is designed for Alice and Bob to compute the LCS of strings S_A and S_B, separately owned by them. PPLS is consisted of three phases. In the first phase, PPAM is used to discover common short sub-strings. In the second phase, some (protected) position information of matched short sub-strings is revealed to each other. Finally, longer common sub-strings can be discovered, until the longest common sub-string is found.

In the first phase Alice and Bob agree upon a parameter l, so that both S_A and S_B are broken into two sets of l-grams $L(S_A)$ and $L(S_B)$, respectively. This implies that n-grams whose lengths shorter than l are not considered. In the second phase, positions of matched l-grams are privately computed for discovery of all possible common sub-strings, without explicit unveiling of these positions. These positions can be compromised under two conditions: (1) multiple unlinked shorter sub-strings of a user could be used to infer a longer sub-string of the other user, or (2) vice versa. We eliminate these problems by adding randomly generated "faked sub-strings" (or, faked ECC points in our design), such that the protected sub-strings cannot be exhaustively matched by other unlinked shorter sub-strings or a longer sub-string. A randomization function $\boldsymbol{R}$ is responsible for generation of those faked points. In the mean time, the randomization should still allow Alice and Bob to discover their adjacent matched l-grams within a few rounds of interactions. Selection on the number of faked points is based on the balance between protection of location information and computing cost. In the third phase, Alice (Bob) generates all possible sub-strings based on her (his) common, sequenced l-grams. These sub-strings are used for matching by PPAM to discover the longest common sub-string. Before we introduce more details about the three phases several observations are presented 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.

In the first phase, Alice and Bob determine a set of matched l-grams $L(C) = \{c_1, c_2, \ldots c_w\}$ between $L(S_A)$ and $L(S_B)$ using PPAM. In the second phase Alice (Bob) first enumerates all sub-strings, $U_A = \{\alpha_1, \alpha_2, \ldots, \alpha_{n'}\}$ ($U_B = \{\beta_1, \beta_2, \beta_3, \ldots, \beta_{m'}\}$) that can be generated by concatenation of some adjacent c_k in $L_A(L_B)$. Any record in $U_A(U_B)$ is removed if it is a sub-string of some other longer string in $U_A(U_B)$. Next, $\forall \alpha_i \in \{\alpha_1, \alpha_2, \ldots, \alpha_{n'}\}$, Alice generates l-grams $L(\alpha_i) = \{a_{i1}, a_{i2}, \ldots\}$. The set of all the l-grams generated based on U_A is denoted as $L(U_A) = \{L(\alpha_1), L(\alpha_2), \ldots, L(\alpha_{n'})\}$. For each $L(\alpha_i) = \{a_{i1}, a_{i2}, \ldots\} \in L(U_A)$, Alice invokes $\boldsymbol{E}$ of PPAM to get $\boldsymbol{E}(L(\alpha_i)) = \{\boldsymbol{E}(a_{i1}), \boldsymbol{E}(a_{i2}), \ldots\}$. Then $\forall \boldsymbol{E}(L(\alpha_i)) \in \{\boldsymbol{E}(L(\alpha_1)), \boldsymbol{E}(L(\alpha_2)), \ldots, \boldsymbol{E}(L(\alpha_{n'}))\}$, Alice generates randomized-shuffled ciphertext $\boldsymbol{R}(\boldsymbol{E}(L(\alpha_i)))$, by inserting random ECC points to $\boldsymbol{E}(L(\alpha_i))$ and then shuffling their order.

The set of randomized-shuffled ciphertexts is denoted as $\boldsymbol{R}(\boldsymbol{E}(L(U_A)) = \{\boldsymbol{R}(\boldsymbol{E}(L(\alpha_1))), \boldsymbol{R}(\boldsymbol{E}(L(\alpha_2))), \ldots, \boldsymbol{R}(\boldsymbol{E}(L(\alpha_{n'})))\}$. Then Alice transforms it into $\boldsymbol{R}(\boldsymbol{E}(L(U_A))'$ by inserting an *infinite point O* of $\mathcal{E}$ between every two entries in $\boldsymbol{R}(\boldsymbol{E}(L(U_A))$ (i.e., use O as a symbol to separate every two entries in $\boldsymbol{R}(\boldsymbol{E}(L(U_A)))$. Following a similar procedure, Bob also generates $L(U_B) = \{L(\beta_1), L(\beta_2), \ldots, L(\beta_{m'})\}$ and $\boldsymbol{R}(\boldsymbol{E}(L(U_B))'$. Alice and Bob then exchange $\boldsymbol{R}(\boldsymbol{E}(L(U_A))'$ and $\boldsymbol{R}(\boldsymbol{E}(L(U_B))'$.

With $L(U_A)$ and $\boldsymbol{R}(\boldsymbol{E}(L(U_B))'$ as its inputs, Alice generates $\boldsymbol{M}\{L(U_A), \boldsymbol{R}(\boldsymbol{E}(L(U_B))'\}$ by the modification function $\boldsymbol{M}$ of PPAM. In the process, the randomization and shuffling and insertion functions mentioned above are also employed. Bob also generates $\boldsymbol{M}\{L(U_B), \boldsymbol{R}(\boldsymbol{E}(L(U_A))'\}$ following a similar procedure. Alice and Bob then exchange their modified ciphertexts $\boldsymbol{M}\{L(U_A), \boldsymbol{R}(\boldsymbol{E}(L(U_B))'\}$ and $\boldsymbol{M}\{L(U_B), \boldsymbol{R}(\boldsymbol{E}(L(U_A))'\}$. Then Alice (Bob) can use $\boldsymbol{D}$ to find $L(U_C)$, which denotes the common l-grams between $L(U_A)$ and $L(U_B)$ with adjacent positions in $U_A(U_B)$. In the last phase, Alice (Bob) generates all possible sub-strings based on

$L(U_C)$ *and* $U_A(U_B)$. These sub-strings are used for matching by PPAM to discover the longest common sub-string.

During execution of $\boldsymbol{E}$ and $\boldsymbol{M}$, Alice/Bob can protect location information of ciphertexts or modified-ciphertexts by insertion of random ECC points generated by $\boldsymbol{R}$. For example, in PPAM when Alice encrypts A to get $\boldsymbol{E}(A) = [E(a_0), E(a_1),\ldots, E(a_n)]$, she could call $\boldsymbol{R}$ to generate additional $2s$ points and attach them with the real ciphertexts, i.e., $\boldsymbol{R}(\boldsymbol{E}(A)) = [E(a_0), E(a_1),\ldots, E(a_n), E(a'_0), E(a'_1),\ldots, E(a'_s)]$, where s is a chosen large integer, $\forall\, i \in [0, s]$, $E(a'_i) = [E(a'_i)_f, E(a'_i)_b] = (r_AQ, P_i)$, P_i is a randomly generated faked point. $\boldsymbol{R}$ guarantees that Bob cannot tell whether or not a point represents a real attribute record, or a faked point. The detail of PPLS is introduced as follows.

Protocol II: Privacy-Preserving LCS Computation

Input: Alice's string S_A*, Bob's string* S_B*, shortest sub-string length l, and all other security parameters presented in PPAM*

Output: the longest common sub-string LCS between S_A *and* S_B

Phase 1

1. Alice: *generate* $L(S_A)$ *from* S_A
   Bob: *generate* $L(S_B)$ *from* S_B
2. Alice ↔ Bob: *Alice and Bob implement PPAM to find* $L(C) = \{c_1, c_2, \ldots c_w\}$*, the matched reocrds of* $L(S_A)$ *and* $L(S_B)$

Phase 2

3. Alice: *generate plaintexts* $L(U_A)$ *and ciphertexts* $\boldsymbol{R}(\boldsymbol{E}(L(U_A))'$ *by taking the following steps:*
   (a) *generate sub-strings* $U_A = \{\alpha_1, \alpha_2, \ldots, \alpha_{n'}\}$, $\forall\, \alpha_i \in U_A$, α_i *is generated from the concatenation of some* $c_{a0}, c_{a1}, \ldots, \in L(C)$*, where* $c_{a0}, c_{a1}, \ldots$ *are adjacent in* S_A
   (b) $\forall\, \alpha_i \in U_A$, α_i *is removed from* U_A *if there is another* $\alpha_i' \in U_A$ *,*α_i *is a substring of* α_i'
   (c) *for* $\alpha_i = \alpha_1$ *to* $\alpha_i = \alpha_{n'}$*, generate l-grams* $L(\alpha_i) = \{a_{i1}, a_{i2}, \ldots\}$*, the set of all the l-grams of* U_A *is denoted by* $L(U_A) = \{L(\alpha_i), L(\alpha_2), \ldots, L(\alpha_{n'})\}$
   (d) *for* $L(\alpha_i) = L(\alpha_1)$ *to* $L(\alpha_i) = L(\alpha_{n'})'$ *suppose* $L(\alpha_i) = a_{i1}, a_{i2},\ldots,$ *, encrypt* $a_{i1}, a_{i2},\ldots$ *to generate* $\boldsymbol{E}(a_{i1}), \boldsymbol{E}(a_{i2}),\ldots$ *to get* $\boldsymbol{E}(L(\alpha_i)) = \{\boldsymbol{E}(a_{i1}), \boldsymbol{E}(a_{i2})\ldots\}$
   (e) *for* $\boldsymbol{E}(L(\alpha_i)) = \boldsymbol{E}(L(\alpha_1))$ *to* $\boldsymbol{E}(L(\alpha_i)) = \boldsymbol{E}(L(\alpha_{n'}))$*, call* $\boldsymbol{R}$ *to add random points into* $\boldsymbol{E}(L(\alpha_i))$ *and then shuffle the order of entries in* $\boldsymbol{E}(L(\alpha_i))$*, the randomized-shuffled ciphertexts is denoted by* $\boldsymbol{R}(\boldsymbol{E}(L(\alpha_i))$
   (f) *generate* $\boldsymbol{R}(\boldsymbol{E}(L(U_A)) = \{\boldsymbol{R}(\boldsymbol{E}(L(\alpha_1)), \boldsymbol{R}(\boldsymbol{E}(L(\alpha_2)), \ldots, \boldsymbol{R}(\boldsymbol{E}(L(\alpha_{n'}))\}$ *and transform it into* $\boldsymbol{R}(\boldsymbol{E}(L(U_A))'$ *by inserting an infinite ECC point O between every two entries in* $\boldsymbol{R}(\boldsymbol{E}(L(U_A))$
   Bob: *generate* $L(U_B)$ *and* $\boldsymbol{R}(\boldsymbol{E}(L(U_B))'$ *following the same steps as Alice*
4. Alice ↔ Bob: *exchange* $\boldsymbol{R}(\boldsymbol{E}(L(U_A))'$ *and* $\boldsymbol{R}(\boldsymbol{E}(L(U_B))'$
5. Alice: *compute* $\boldsymbol{M}(L(U_A), \boldsymbol{R}(\boldsymbol{E}(L(U_B))')$*, invoke* $\boldsymbol{R}$ *and shuffling and insertion function during the modification process, send* $\boldsymbol{M}(L(U_A), \boldsymbol{R}(\boldsymbol{E}(L(U_B))')$ *to Bob*
   Bob: *compute* $\boldsymbol{M}(L(U_B), \boldsymbol{R}(\boldsymbol{E}(L(U_A))')$ *and send it to Alice following the same procedures*
6. Alice: *compute* $\boldsymbol{D}(\boldsymbol{M}(L(U_A), \boldsymbol{R}(\boldsymbol{E}(L(U_B))'))$*, discover all matched l-grams* $L(U_C) = \{c_0', c_1', \ldots\}$ *and record possible position information, i.e., for any* $c_i, c_i' \in L(U_C)$*, (a)* $c_i, c_i' \in L(\alpha_i)$ & $c_i, c_i' \in \boldsymbol{R}(\boldsymbol{E}(L(\beta_j))$*, where* $L(\alpha_i) \in L(U_A)$ *and* $\boldsymbol{R}(\boldsymbol{E}(L(\beta_j)) \in \boldsymbol{R}(\boldsymbol{E}(L(U_B))'$*; (b)* c_i, c_i' *have adjacent positions in* α_i
   Bob: *discover matched l-grams using a similar confirmation test*

Phase 3

7. Alice: *generate possible sub-strings based on the common, sequenced l-grams in* $L(U_C)$*, execute PPAM with Bob to find the matched sub-string with the longest length*
   Bob: *find the matched sub-string with the longest length following the same procedures*

The crypto computing complexity of PPLS is that of PPAM plus the additional costs required for randomization and shuffling. Both the parameter l and the length of the LCS are important parameters that affect the running time of PPLS. For example, if two input strings have no LCS, then only Phase 1 will need to be run, i.e., only one round of matching between two sets of l-grams is required. On the other hand, when two input string have some identical substrings, it is necessary to devise a proportional number of steps for shuffling and randomization (faked points).

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 large random numbers to multiply records to generate *encrypted records*. They are then associated with a fixed ECC point, e.g., $(r_{A1}a_i + r_{A2})Q_A$. We claim that record a_i cannot be deduced from $(r_{A1}a_i + r_{A2})Q_A$ by any passive attack because r_{A1} and r_{A2} are large enough to guarantee the ECDLP. In [1] attributes were mapped to the roots of an equation and the coefficients of the equations were encrypted by a homormorphic encryption scheme and were sent to the other user for evaluation. Because in our design all ciphertexts are decrypted by using the homomorphic encryption scheme, same as the design proposed 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 first part of the proof is similar to the proof of Attribute Confidentiality. For the second part, we assume that Alice calls $\boldsymbol{R}$ to add faked points into $\boldsymbol{E}(A)$, she gets $\boldsymbol{R}(\boldsymbol{E}(A_i)) = [E(a_1), E(a_2),\ldots, E(a_n), E(a'_1), E(a'_2),\ldots, E(a'_s)]$. Suppose each record in A represents a l-gram. Since each $E(a'_i) = [E(a'_i)_f, E(a'_i)_b] = (r_A Q, P_i)$ represents a faked encrypted l-gram, where P_i is a randomly generated faked point, even if the first n real l-grams are matched, the adversary only has a small probability to discover the corresponding sub-string by combining them in right sequence from the whole $(s + n)$ l-grams. 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 protocol Alice needs to map each record to one root of a polynomial equation, encrypts the coefficients of the equation through a homomorphic encryption scheme and sends the ciphertexts to Bob. Bob then can evaluate the polynomial based on the encrypted coefficients and his own records. Then the evaluated polynomial is sent back to Alice such that she can determine the matching outcome. The process needs $O(n^2)$ homomorphic encryptions (e.g., point multiplication in ECC) for Bob to evaluate the equation. That is, for each input x (record), Bob needs to evaluate the whole n-term polynomial equation $a_0 + a_1 \otimes x + a_2 \otimes x^2 + \ldots + a_n \otimes x^n$ through n times operation of $\otimes$, which denotes the operation of a homomorphic encryption. 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/