Breaking two PSI-CA protocols in polynomial time

Yang Tan[1][0000−0003−1015−5279] and Bo Lv[2]

[1] Xinxindigits, Qianhai, Shenzhen, Guangdong, China
t.yang03@mail.scut.edu.cn
[2] Huizhou University, Huizhou, Guangdong, China
lvbo@hzu.edu.cn

Abstract. Private Set Intersection Cardinality(PSI-CA) is a type of secure two-party computation. It enables two parties, each holding a private set, to jointly compute the cardinality of their intersection without revealing any other private information about their respective sets.
In this paper, we manage to break two PSI-CA protocols by recovering the specific intersection items in polynomial time. Among them, the PSI-CA protocol proposed by De Cristofaro et al. in 2012 is the most popular PSI-CA protocol based on the Google Scholar search results and it is still deemed one of the most efficient PSI-CA protocols.
In this paper, we also propose several solutions to these protocols' security problems.

Keywords: PSI-CA · PSI · DDH · Bloom Filter.

1 Introduction

1.1 A Subsection Sample

Private Set Intersection Cardinality (PSI-CA) is an important primitive of secure two-party computation. It enables two parties, each holding a private set, to jointly compute the cardinality of their intersection without revealing any other private information about their respective sets.

PSI-CA is a useful tool for privacy protection. For example, on social media (such as Facebook, WhatsApp), two users can determine whether they should become friends based on the number of common connections without leaking any private contact. Other applications include DNA sequence similarity comparison, anonymous authentication, etc. Moreover, it has wide applications in Federated Learning where parties can train a machine learning model together without sharing their private data. In Federated Learning, PSI/PSI-CA protocol is used as a basic tool to determine participants overlapped samples or the cardinality of these samples without leaking any other information.

In the existing PSI-CA protocols, the most efficient ones require linear computation and communication complexity. Early PSI-CA protocols are usually based on Public Key Cryptography [1, 7]. Recent works [9, 10, 12, 15] have tried

to introduce more mechanisms such as bloom filters, homomorphic encryption and oblivious transfer to improve efficiency, but their complexity levels are still linear. Dong and Loukides [11] developed an approximate PSI-CA protocol based on multiple Flajolet-Martin(FM) sketches (space efficient data structure for cardinality estimation). With some sacrifice on accuracy, it claims it can achieve logarithmic complexity.

In this paper, we will try to break two PSI-CA protocols [7,8] in polynomial time. Both of their securities are based on the DDH assumption. Among them, [7] is the most popular PSI-CA protocol based on the Google Scholar search results.

The content of this paper is as follows: Firstly, we give a background introduction. Secondly, we describe some basic definitions involved in this paper. Thirdly, we describe the two targeted PSI-CA protocols and how to break them in polynomial time. Fourthly, we run some attack simulations. Fifthly, we propose some solutions to their security problems. Finally, we make a conclusion.

2 Definitions

In this section, we introduce some basic definitions involved in this paper.

Definition 1. ***Private Set Intersection Cardinality(PSI-CA).*** *A protocol involving Server, on input a set of w items $Y = \{y_1, ..., y_w\}$, and Client, on input a set of v items $X = \{x_1, ..., x_v\}$. It outputs I, where: $I = |X \cap Y|$.*

For PSI-CA protocol, the following privacy requirements should be met:

- **Server Privacy.** Client learns no information beyond: (1) cardinality of set intersection and (2) upper bound on the size of Y.
- **Client Privacy.** No information is leaked about client set X, except an upper bound on its size.
- **Unlinkability.** Neither party can determine if any two instances of the protocol are related, i.e., executed on the same input by Client or Server, unless this can be inferred from the actual protocol output.

There are two common types of security models for general secure multi-party computation:

- **Semi-honest Model.** The adversary in this model is called a semi-honest adversary. It follows the protocol. However, it will try to gain extra information out of the protocol.
- **Malicious Model.** In this model, the adversary is called a malicious adversary. It will try to gain extra information by any kind of method, for example, changing the inputs and outputs, aborting the protocol, etc.

Other notations:

- Two hash functions modeled as random oracles, $H1 : \{0,1\}* \rightarrow G$ in which G is a cyclic subgroup of Z_p of order q and $H2 : \{0,1\}* \rightarrow \{0,1\}^k$ given the security parameter k.

- Two random permutations:$\prod$, $\prod'$.
- Throughout the paper, the notations $a \longleftarrow A, x \leftharpoondown X$ are, respectively, used to represent a is the output of the procedure A, variable x is chosen uniformly at random from set X.

Definition 2. ***Discrete Logarithm Assumption.*** *Let G be a cyclic group and g be its generator. The discrete logarithm problem(DLP) is called (t, ε) hard relative to G if for all algorithms A runs in time t there exists a negligible function ε of security parameter k such that*

$$Pr[\mathcal{A}(g, g^a) = a] \leq \varepsilon$$

Definition 3. ***DDH assumption.*** *Let $\boldsymbol{G}$ be a cyclic group of order q and g is its generator and l is the bit-length of the group size. The following two distributions are computationally indistinguishable*

(1)g^x, g^y, g^{xy}

(2)g^x, g^y, g^z

given x, y, z are randomly and independently chosen from Z_q. To put it in a more formal way, DDH problem is (t, ε) hard if for all algorithms A runs in time t there exists a negligible function ε.

$$|\Pr[x, y \leftarrow \{0,1\}^l : A(g, g^x, g^y, g^{xy}) = 1] - \Pr[x, y \leftarrow \{0,1\}^l : A(g, g^x, g^y, g^z) = 1]| \leq \varepsilon$$

Definition 4. ***Bloom Filter.*** *A Bloom filter(BF) [2] is a data structure that represents a set $X = \{x_1, ..., x_v\}$ by an array of m bits and use k independent uniform hash functions: $\{h_1, h_2, ..., h_k\}$ where $h_i : \{0,1\}* \rightarrow m$. These hash functions are used to insert elements or check the existence of the elements in Bloom Filter.*

A bloom filter may render false positives, i.e., an element not in set X may pass its Bloom Filter's membership test. In practice, we need to adjust the parameters: k, m [5] to make the false positive rate stays less than a certain value(e.g. 2^{-80}).

In this paper, we use a variant of the Bloom Filter(this Inverted Bloom filter can be referred to as **IBF**) which includes the following operations:

- **Initialization** Set all the bits of the m-bit array of IBF to 1.
- **Add(x)** To insert an element $x \in X$ into the Bloom filter, calculate the following k indices: $\{h_1(x), ..., h_k(x)\}$, and set the bits with positions in these indices to 0. Repeat this process till all the members in X are inserted into the Bloom Filter. Then the Bloom Filter of X is represented as $IBF_X \in \{0,1\}^m$.
- **Check(x')** to check whether $x' \in X$, calculate the following k indices: $\{h_1(x'), ..., h_k(x')\}$ and check IBF_X' bits with positions in these indices. If they are all 0, x' is probably in X, otherwise, x' is not in X.

3 Break the PSI-CA protocol from [7]

In 2012, De Cristofaro et al. proposed an efficient PSI-CA protocol with linear complexity in computation and communication [7].

In this article, the authors claimed their PSI-CA protocol was secure under the DDH assumption [3] in the random oracle model against semi-honest adversaries.

However, the original protocol made a small mistake: the authors seem to forget to map the hashes of the private inputs to the chosen cyclic group G. Without this operation, DDH assumption won't stand and computations like stripping off an exponent R_c by exponentiating with R_c's inverse modular q won't work.

This mistake is pointed out and corrected in [18]. The corrections to this protocol are made by mapping the hashes of the private inputs to the chosen cyclic group.

3.1 Description of the Protocol.

In Table 1, we present you this protocol.

Table 1. PSI-CA Protocol from [7].

Client, on input $C = \{c_1, ..., c_v\}$	Common inputs $p, q, g, H1, H2$	Server, on input $S = \{s_1, ..., s_w\}$
$R_c \leftarrow Z_q$		$(\hat{s_1}, ..., \hat{s_w}) \leftarrow \prod(S)$
$\forall i\ 1 \leq i \leq v:$		$\forall j\ 1 \leq j \leq w:$
$hc_i = H1(c_i)$		$hs_j = H1(\hat{s_j})$
$a_i = (hc_i)^{R_c}$		
	$\xrightarrow{a_1,...,a_v}$	
		$R_s \leftarrow Z_q$
		$\forall i\ 1 \leq i \leq v: a_i^{'} = (a_i)^{R_s}$
		$(a_{l_1}^{'}, ..., a_{l_v}^{'}) = \prod{}'(a_i^{'}, ..., a_v^{'})$
		$\forall j\ 1 \leq j \leq w:$
		$bs_j = (hs_j)^{R_s};$
		$\forall j\ 1 \leq j \leq w: ts_j = H2(bs_j)$
	$\xleftarrow[\{ts_1,...,ts_w\}]{\{a_{l_1}^{'},...,a_{l_v}^{'}\}}$	
$\forall i\ 1 \leq i \leq v:$		
$bc_i = (a_{l_i}^{'})^{1/R_c\ mod\ q}$		
$\forall i\ 1 \leq i \leq v:$		
$tc_i = H2(bc_i)$		
Output: $\lvert\{ts_1, ..., ts_w\} \bigcap \{tc_1, ..., tc_v\}\rvert$		

One thing to note, the original paper made a little mistake by simply defining $H1 : \{0,1\}* \rightarrow Z_p$. For this protocol to work, $H1$ should map the private input

to a random element in the cyclic subgroup G. This mistake is also pointed out in [18].

In Table 1, the common inputs p, q are prime numbers. All computations are mod p and q is G's order.

The intuition behind this protocol is straightforward. In the end, all the private inputs go through the same computations: $x \rightarrow \mathrm{H2}(\mathrm{H1}(x)^{R_s})$. If two input numbers are equal, their outputs should be equal as well. In this case, the cardinality of the private intersection pluses one.

Also, due to the DL and DDH assumptions, the Client and the Server can't recover any private input from the intermediate results.

Furthermore, to prevent the Client from recovering intersection set by data's order, this protocol applies two random permutations $\prod$, $\prod'$ to shuffle on the Server side.

3.2 Correctness.

For any c_i held by Client and s_j held by Server, if $c_i = s_j$, hence, $hc_i = hs_j$, we obtain:

$$\begin{aligned} |\{ts_1, \ldots, ts_w\} \cap \{tc_1, \ldots, tc_v\}| &\rightarrow |\{bs_1, \ldots, bs_w\} \cap \{bc_1, \ldots, bc_v\}| \\ &\rightarrow |\{(hs_1)^{R_s}, \ldots, (hs_w)^{R_s}\} \cap \{(hc_1)^{R_s}, \ldots, (hc_v)^{R_s}\}| \\ &\rightarrow |\{hs_1, \ldots, hs_w\} \cap \{hc_1, \ldots, hc_v\}| \\ &\rightarrow |S \cap C| \end{aligned} \tag{1}$$

3.3 Recovering Intersection Items in Polynomial time

Idea Behind the Attack It is true that due to the hardness of DLP, without the knowledge of R_s, a'_{l_i} is indistinguishable from a random value and the Client adversary can't tell a'_{l_i} comes from which a_i. It looks like $\{bc_i\}$ is as far as the adversary can get. Also, because of the random permutations, the Client can't recover the intersection set by items' order.

However, the authors omit one fact: an invariable R_s is involved in process of generating tc_i and ts_j. If the Client adversary can recover something invariable related to R_s from the intersection of $\{tc_i\}$ and $\{ts_j\}$, and link it to $\{hc_i\}$, it is possible to recover the private intersection set.

Simulations and Instantiations of the Random Oracles In the security proof part (section 4.1, Server Privacy) of [7], it defines a simulator $\overline{SIM}$ which interacts with a distinguisher D. $\overline{SIM}$ simulates random oracles $H1$ and $H2$. $\overline{SIM}$ responds to $H1(x)$ queries from D with g^{rh_i} for a random $rh_i \in Z_q$, and stores (x, g^{rh_i}) in table T_H for consistency and to queries $H2(x)$ with a random string, using $T_{H'}$ to store queries-response for consistency.

In the ideal world, the random oracle could be performed like a trusted third-party or a black-box. Client can simply issue queries and get responses from $H1$. It has no access to the random rh_i, nor the Table T_H.

However, the random oracle is an imaginary object, to implement the protocol [7] in real world, it has to be instantiated. In the instantiation of a random oracle like $H1$, a random number rh_i has to be generated for every distinct query. In common practice, it uses a concrete, possibly keyed, cryptographic hash function, such as SHA functions, or hash functions constructed by utilizing indistinguishability obfuscation [14].

In this paper, we just assume rh_i is already random and we focus on whether Client has access to rh_i. For the following attack to work, the Client needs access to this random value. In practice, to implement this PSI-CA protocol in the two-party setting, the Client has to generate rh_i on its side. It couldn't issue $H1$ queries to the Server or other parties. Otherwise, private inputs will be revealed. Ergo, we could assume the Client generates the random value rh_i by itself and has access to it.

How to Perform the Attack In the following parts of this section, we will show you how to recover an invariable from the intersection of $\{tc_i\}$ and $\{ts_j\}$ and based on this invariable, recover the private intersection set in polynomial time.

- Assume the intersection cardinality is k, and the intersection of $\{tc_i\}$ and $\{ts_j\}$ is $\{tc_{f_1}, ..., tc_{f_k}\}$.
- Pick one value from the intersection: tc_{f_i}, Client can get the corresponding bc_{f_i}. Because of the random shuffle on the Server side, we assume bc_{f_i} comes from hc_m and $hc_m = g^{rh_m}$, then we have: $bc_{f_i} = (a'_{l_{f_i}})^{1/R_c \bmod q} = ({hc_m}^{R_c \cdot R_s})^{1/R_c \bmod q} = (hc_m)^{R_s} = g^{rh_m \cdot R_s}$ where $1 \leq m \leq v$.
- With $hc_j = g^{rh_j}$, Client can perform an exhaustive search by computing: ${bc_{f_i}}^{1/rh_j \bmod q} = g^{rh_m \cdot R_s \cdot 1/rh_j \bmod q} = g^{rh_m/rh_j \cdot R_s \bmod q}$ for $\forall j \ 1 \leq j \leq v$. When $j = m$, the result of this computation is exactly g^{R_s} and g^{R_s} is the invariable we try to recover.
- For every element tc_{f_i} in $\{tc_{f_1}, ..., tc_{f_k}\}$, repeat the above exhaustive search on its corresponding bc_{f_i}, and record the one $hc_j = g^{rh_j}$ that can make ${bc_{f_i}}^{1/rh_j \bmod q}$ equals g^{R_s}.
- In the end, $\{hc_j\}$'s corresponding $\{c_j\}$ is the intersection set the Client adversary tries to recover. Ergo, by the definition of PSI-CA protocol, this protocol is compromised.

One thing to note, before the attack, the Client adversary doesn't know the exact value of g^{R_s} in advance, so it has to repeat the exhaustive search at least twice to determine the value of g^{R_s}.

Complexity Analysis Assume the intersection cardinality is k, one time exhaustive search for an intersection element needs v exponentiations. Thereby, the

overall complexity of recovering all the intersection elements is $O(k \times v)$ which is a quadratic polynomial. Consequently, we break the protocol in polynomial time.

4 Break the PSI-CA protocol from [8]

In [8], the authors proposed two PSI-CA protocols in the presence of malicious adversaries: one with Bloom Filter [2], one without Bloom Filter.

Both protocols are based on the DDH assumption [4] and the protocol with Bloom Filter is more efficient in communication.

In both protocols, Zero-Knowledge Proof(ZKP) is used to prevent Malicious adversaries from misbehaving. ZKP can be viewed as some extra processes of challenge, response, and verification to make sure that all the intermediate results are not tampered by a malicious adversary. Since our attack targets the protocols themselves and has nothing to do with whether the intermediate results be tampered or not, for the simplicity of the illustration of the attack, we will leave out the ZKP parts in the description of the Protocol.

In this section, we will show the attack against the PSI-CA protocol without Bloom Filter, and the attack against the PSI-CA protocol with Bloom Filter is nearly the same.

4.1 ElGamal Encryption

This protocol uses ElGamal encryption [13] to encrypt Client's private inputs.

The ElGamal encryption is multiplicative homomorphic and it is defined as follows:

- Setup (1^κ) - On security parameter input 1^κ, a trusted authority outputs a public parameter $par = (p, q, g)$,where p, q are prime numbers such that q divides $p-1$ and g is a generator of the cyclic subgroup G of Z_p of order q.

- KeyGen (par) - User U chooses $x \leftarrow\!\!\!\prec Z_q$, computes $h = g^x$, reveals $pk_U = h$ as his public key and keeps secret $sk_U = x$ to himself.

- Enc (m, pk_U, par, r) - The encryptor encrypts a message $m \in G$ using the public key $pk_U = h$ by computing ciphertext tuple $E_{pk_U}(m) = (\alpha, \beta) = (g^r, mh^r)$, where $r \leftarrow\!\!\!\prec Z_q$.

- Dec $(E_{pk_U}(m), sk_U)$ - On receiving ciphertext tuple $E_{pk_U}(m) = (\alpha, \beta) = (g^r, mh^r)$, the decryptor U decrypts it using the secret key $sk_U = x$ by computing $\frac{\beta}{\alpha^x} = \frac{m(g^x)^r}{(g^r)^x} = m$.
  The ElGamal encryption is semantically secure provided the DDH problem is hard in G.

4.2 Description of the Protocol.

The PSI-CA Protocol without Bloom Filter is shown in Table 2.

Table 2. PSI-CA Protocol from [8].

Client, on input $X = \{x_1, \cdots, x_v\}$	Public: (p, q, g)	Server, on input $Y = \{y_1, \ldots, y_w\}$
$(pk_C, sk_C) \leftarrow$ KeyGen(par) (i) chooses $r_{x_1}, \ldots, r_{x_v} \leftarrowtail Z_q$, $E_{pk_C}(x_i) = (c_{x_i}, d_{x_i}) = (g^{r_{x_i}}, x_i h^{r_{x_i}})$ $(ii) R_1 = \{E_{pk_C}(x_1), \ldots, E_{pk_C}(x_v)\}$	$\xrightarrow{R_1}$	
		(i)$r \leftarrowtail Z_q$, $\widehat{Y} = \{t_1 = (y_1)^r, \ldots, t_w = (y_w)^r\}$ (ii) for $i = 1, \ldots, v$, $(E_{pk_C}(x_i))^r = (\hat{c}_{x_i}, \hat{d}_{x_i}) = (c_{x_i}^r, d_{x_i}^r)$ (iii)$\bar{X} =$ Perm $\{(E_{pk_C}(x_1))^r, \ldots, (E_{pk_C}(x_v))^r\}$ $= \{(E_{pk_C}(\bar{x}_1))^r, \ldots, (E_{pk_C}(\bar{x}_v))^r\}$; (iv)$R_2 = \left\langle \widehat{Y} = \{t_1, \ldots, t_w\}, \bar{X} \right\rangle$
	$\xleftarrow{R_2}$	
For $i = 1, \ldots, v$, $s_i = (\bar{x}_i)^r \leftarrow \mathrm{Dec}\left((E_{pk_C}(\bar{x}_i))^r, sk_C\right)$; $\lvert X \cap Y \rvert = \lvert \{s_1, \ldots, s_v\} \cap \{t_1, \ldots, t_w\} \rvert$		

4.3 Intuition

From Table 2, we can see that the Client encrypts his private data set $X = \{x_1, ..., x_v\}$ with ElGamal encryption and then sends it to Server. Because ElGamal encryption is multiplicative homomorphic, the Server can exponentiate these encrypted values with its private input r and generate:

$$(E_{pk_C}(x_i))^r = (\hat{c}_{x_i}, \hat{d}_{x_i}) = (c_{x_i}^r, d_{x_i}^r),\ i = 1, \ldots, v \tag{2}$$

Server randomly shuffles them and sends them back to Client. Client can decrypt these values and get:

$$s_i = (\bar{x}_i)^r \leftarrow \mathrm{Dec}((E_{pk_C}(\bar{x}_i))^r, sk_C) \tag{3}$$

Server also encrypts its own private data set $Y = \{y_1, ..., y_w\}$ by exponentiating them with exponent r and generates:

$$\widehat{Y} = \{t_1 = (y_1)^r, \ldots, t_w = (y_w)^r\} \tag{4}$$

and sends $\widehat{Y}$ to the Client.

In the end, the Client gets all private inputs' exponentiations with r as the exponent. Thereby, the Client can get the intersection cardinality by $|\{s_1, \ldots, s_v\} \cap \{t_1, \ldots, t_w\}|$. Also, because of the DL and DDH assumptions, Client and Server can't recover any private input from the intermediate results.

Furthermore, similar to the previous PSI-CA protocol, because of the random permutation: Perm, the Client adversary can't recover the intersection set by data's order.

4.4 Recovering Intersection Items in Polynomial time

The Idea Behind the Attack This protocol made the same mistakes as the previous PSI-CA protocol.

In the processes of generating $\{s_1, \ldots, s_v\}$ and $\{t_1, \ldots, t_w\}$, an invariable r introduced by Server is involved.

Even though the Client can't recover $\bar{x}_i$ from s_i due to the hardness of the DLP, if the Client adversary can recover something invariable related to r from the intermediate results and link this invariable to the private input, it's possible to recover the intersection set.

How to Perform the Attack Assume the intersection cardinality is k, then we have $\{s_1, \ldots, s_v\} \cap \{t_1, \ldots, t_w\} = \{s_{l1}, \ldots, s_{lk}\}$.

Now, let's try to recover an invariable from s_{li}'s corresponding $(E_{pk_C}(\bar{x}_{li}))^r$ for $i = 1, ..., k$.

- Extract $(c_{\bar{x}_{li}})^r$ from $(E_{pk_C}(\bar{x}_{li}))^r = ((c_{\bar{x}_{li}})^r, (d_{\bar{x}_{li}})^r)$, and $(c_{\bar{x}_{li}})^r = (g^{r_{\bar{x}_{li}}})^r = g^{r \times r_{\bar{x}_{li}}}$.
- Client adversary performs an exhaustive search attack on $(c_{\bar{x}_{li}})^r$ by computing: $((c_{\bar{x}_{li}})^r)^{1/r_{x_j} \bmod q} = (g^{r_{\bar{x}_{li}}/r_{x_j} \bmod q})^r$ for $\forall j \ 1 \leq j \leq v$. When $r_{\bar{x}_{li}} = r_{x_j}$, the result is exactly g^r.
- As r_{x_j} has a one-to-one mapping relation with x_j, the Client can be certain that $x_j \in X \cap Y$.
- Client repeat this process for $\forall i \ 1 \leq i \leq k$ till the whole intersection set is recovered.

Similar to the previous attack, the Client adversary doesn't know the exact value of g^r in advance. It has to repeat the exhaustive search at least 2 times to determine this value.

As to the PSI-CA protocol with Bloom filter, its main difference from the PSI-CA protocol without Bloom filter is: Instead of transmitting $\hat{Y}$, it transmits the bloom filter of $\hat{Y}$ to reduce the communication costs. The intersection cardinality can be computed by counting the number of s_i that passes the membership test of $\hat{Y}$'s Bloom Filter.

Obviously, this new construction won't affect us performing the same attack: using s_{li}'s corresponding $(E_{pk_C}(\bar{x}_{li}))^r$ to recover invariable g^r provided that s_{li} passes the membership test of $\hat{Y}$'s Bloom Filter. Thereby, we won't elaborate on this protocol's specific construction and the attack against it.

Complexity Analysis Assume the intersection cardinality is k, one time exhaustive search attack for an intersection item needs v inverses modular q and exponentiations. Thereby, the overall complexity of recovering the whole intersection set is $O(k \times v)$ which is a quadratic polynomial. Consequently, we break the protocol in polynomial time.

5 Attack Simulations

In the previous section, we discussed two PSI-CA protocols and how to break them in polynomial time. In this section, we will run some experiments to simulate the attacks.

For the rest of this paper, we will refer to the PSI-CA protocol from [7] as PSI-CA protocol 1 and the PSI-CA protocol from [8] as PSI-CA protocol 2.

In these experiments, we use Python 3.6.8 for protocol implementations and attack simulations. We run these simulations on a Linux Red Hat 4.8.5-44 server with Intel Cure2 Duo T7700(2.4GHz) as CPU.

We choose two sets of parameters from [17] for protocols' Z_p and its cyclic subgroup G: **1024-bit MODP Group with 160-bit Prime Order Subgroup**, **2048-bit MODP Group with 224-bit Prime Order Subgroup**. They have 80-bit and 112-bit security levels, respectively.

For each protocol and each set of parameters, we run three groups of test. In Group 1, we have $v = 1000$, $w = 25000$ and $k = 500$ which means Client's private data set is of size 1000, Server's private data set is of size 2500, and their intersection cardinality is 500. Similarly, in Group 2, we have $v = 2000$, $w = 5000$ and $k = 1000$. In Group 3, we have $v = 10000$, $w = 25000$ and $k = 5000$.

In these attacks, to determine the values of the invariables, we randomly pick two bc_{f_i} for PSI-CA protocol 1 and two $(c_{\bar{x}_{1i}})^r$ for PSI-CA protocol 2 and perform the exhaustive search two times at first. In the end, we record the total time of recovering the intersection set for these protocols.

The experiment results of breaking PSI-CA protocol 1 and PSI-CA protocol 2 are shown in Table 3 and Table 4, respectively.

Table 3. Time of breaking PSI-CA Protocol 1.

	Group 1	Group 2	Group 3
1024-bit Group	63.093s	263.533s	6468.485s
2048-bit Group	328.924s	1312.951s	32808.890s

From these tables, we can see that, these protocols can be broken in a short time period. Furthermore, for both protocols, Group 2 is about 4 times the attack time of Group1, and Group 3 is about 10 times the attack time of Group 1. Results verify the correctness of the complexity analyses of these attacks: quadratic polynomials linear to $k \times v$.

Table 4. Time of breaking PSI-CA Protocol 2.

	Group 1	Group 2	Group 3
1024-bit Group	68.634s	263.834s	6434.539s
2048-bit Group	336.421s	1325.594s	32981.475s

6 Security Solutions

In this section, we focus on the solutions to the security problems of the PSI-CA protocols we broke in previous section.

In our solutions, we don't consider solutions by introducing a semi-honest third party. It would be meaningless since we could easily come up with a much more efficient PSI-CA protocol. For example, in [16], by introducing a semi-honest third-party, they came up with an highly efficient PSI protocol than these protocols in a two-party setting.

Thereby, we will also propose 2 security solutions for each protocol without introducing a third party.

One thing to note, the second security solution of PSI-CA protocol 1 is derived from the intersection size protocol in [1]. It's similar to PSI-CA protocol 1 and can resist our attack. For readers' reference, we point it out and make a few adjustments to make it more similar to PSI-CA protocol 1.

6.1 Two Security Solutions for PSI-CA protocol 1

First Solution The first solution we come up with is to discard the subgroup G with prime order q.

The reason that the Client adversary can recover the invariable g^{R_s} is that in the chosen cyclic subgroup G, the Client can successfully strip off tc_{f_i}'s exponent rh_m by an exhaustive search: ${bc_{f_i}}^{1/rh_j \bmod q} = g^{rh_m \cdot R_s \cdot 1/rh_j \bmod q} = g^{rh_m/rh_j \cdot R_s \bmod q}$ for $\forall j; 1 \leq j \leq v$. When $j = m$, the exponent rh_m is stripped off.

In this solution, we discard the cyclic subgroup G, and simply do all the computations *mod* p. We represent *mod* p as MODP group P. Hash function $H1$ modeled as random oracle is redefined as: $H1 : \{0,1\}* \rightarrow Z_p$. This time, the Client adversary won't be able to strip off the exponent to recover an invariable.

The fixed protocol is shown in Table 5.

According to the Fermat's little theorem [19], group P is of order $p-1$, In this construction, we choose p as a safe prime (both p and $q = (p-1)/2$ are primes) and R_c to be coprime with $p-1$ so that the Client can still strip off exponent R_C to generate bc_i.

How this solution solves the security problem If the Client adversary wants to perform the previous exhaustive search attack to recover an invariable, he has to strip off some exponents. The generator of group P is unknown, we assume it's g_1 and a targeted bc_i comes from c_j, then $bc_i = (a'_{l_i})^{1/R_c \bmod (p-1)} = {a_j}^{R_s} = {hc_j}^{R_s}$.

Table 5. First Security Solution for PSI-CA Protocol 1.

Client, on input $C = \{c_1, ..., c_v\}$		Server, on input $S = \{s_1, ..., s_w\}$
pick $R_c \leftarrow Z_p$		
with $gcd(R_c, p-1) = 1$		$(\hat{s_1}, ..., \hat{s_w}) \leftarrow \prod(S)$
$\forall i\ 1 \le i \le v$:		$\forall j\ 1 \le j \le w$:
$hc_i = H1(c_i)$		$hs_j = H1(\hat{s_j})$;
$a_i = (hc_i)^{R_c}$		
	$\xrightarrow{a_1,...,a_v}$	
		$R_s \leftarrow Z_p$
		$\forall i\ 1 \le i \le v : a_i^{'} = (a_i)^{R_s}$
		$(a_{l_1}^{'}, ..., a_{l_v}^{'}) = \prod{}'(a_i^{'}, ..., a_v^{'})$
		$\forall j\ 1 \le j \le w : bs_j = (hs_j)^{R_s}$;
		$\forall j\ 1 \le j \le w : ts_j = H2(bs_j)$
	$\xleftarrow[\{ts_1,...,ts_w\}]{\{a_{l_1}^{'},...,a_{l_v}^{'}\}}$	
$\forall i\ 1 \le i \le v$:		
$bc_i = (a_{l_i}^{'})^{1/R_c\ mod\ (p-1)}$		
$\forall i\ 1 \le i \le v$:		
$tc_i = H2(bc_i)$		
Output: $\lvert\{ts_1, ..., ts_w\} \bigcap \{tc_1, ..., tc_v\}\rvert$		

Further assume $hc_j = g_1^x$, then $bc_i = g_1^{x \cdot R_s}$. To perform the previous attack to recover an invariable, e.g., $g_1^{R_s}$ or R_s, the Client has to gain access to x and g_1. However, when p is large (e.g. 2048-bit), both x and g_1 are hard to obtain. Even if we assume g_1 is known to the Client, because of the hardness of DLP, x still can't be accessed by the Client adversary. Moreover, since $p-1$ is a composite number, the inverse of x modular $p-1$ which is necessary to strip off the exponent x and extract the invariable, doesn't always exist.

Second Solution We notice the intersection size protocol in [1] is similar to the PSI-CA protocol 1 and it's secure from our attack.

In this construction, it also discards the cyclic subgroup G. Furthermore, it doesn't need to choose R_c to be coprime with $p-1$ since it doesn't need to strip off exponent R_c.

We modify this construction a little bit on some notations and symbols etc to make it more similar to PSI-CA protocol 1. The modified construction is shown in Table 6.

In this construction, hash function $H1$ is also defined as: $H1 : \{0,1\}* \rightarrow Z_p$ and p is a safe prime number.

This construction is correct as long as exponentiations over modular p are commutative, i.e., $(y^{R_s})^{R_c} = (y^{R_c})^{R_s}$.

This construction is also secure from our attack, the reason is the same as our previous construction: In order to recover an invariable, e.g., R_s or $g_1^{R_s}$, the adversary has to break the DLP to strip off some exponents which is infeasible.

Table 6. Second Security Solution for PSI-CA Protocol 1.

Client, on input $C = \{c_1, ..., c_v\}$		Server, on input $S = \{s_1, ..., s_w\}$
$R_c \leftarrow Z_q$ $\forall i\ 1 \le i \le v:$ $hc_i = H1(c_i)$ $a_i = (hc_i)^{R_c}$		$(\hat{s_1}, ..., \hat{s_w}) \leftarrow \prod(S)$ $\forall j\ 1 \le j \le w:$ $hs_j = H1(\hat{s_j})$;
	$\xrightarrow{a_1,...,a_v}$	
		$R_s \leftarrow Z_q$ $\forall i\ 1 \le i \le v: a_i^{'} = (a_i)^{R_s}$ $(a_{l_1}^{'}, ..., a_{l_v}^{'}) = \prod{}'(a_i^{'}, ..., a_v^{'})$ $\forall j\ 1 \le j \le w:$ $bs_j = (hs_j)^{R_s}$;
	$\xleftarrow[\{bs_1,...,bs_w\}]{\{a_{l_1}^{'},...,a_{l_v}^{'}\}}$	
$\forall j\ 1 \le i \le w:$ $ts_j = bs_j^{R_c}$ **Output:** $\left\|\{ts_1, ..., ts_w\} \bigcap \{a_{l_1}^{'}, ..., a_{l_v}^{'}\}\right\|$		

More details of this protocol and its formal security proof can be found in the original paper.

6.2 Two Security Solutions for PSI-CA protocol 2

First Solution The first solution to PSI-CA protocol 2's security problem is to inject randomness to each member of $\bar{X}$.

The original protocol's vulnerability comes from the invariable in each member of $\bar{X}$. If we can inject randomness to each member of $\bar{X}$, the invariable will become variable and there will be no invariable for the Client adversary to recover.

As to how to inject randomness to each member of $\bar{X}$, we introduce the following pair:

$$\sigma_{x_i} = (g^{\sigma_i}, h^{\sigma_i})$$

in which σ_i is randomly chosen from Z_q. This pair could be viewed as ElGamal encryption of 1. Since ElGamal encryption is multiplicative homomorphic, we could multiply this pair by ${x_i}^r$'s cipher-text: $(E_{pk_C}(x_i))^r = (\hat{c}_{x_i} = (c_{x_i})^r, \hat{d}_{x_i} = (d_{x_i})^r)$, where $i = 1, \ldots, v$ and generate $A_i = \left(\hat{c}_{x_i} = g^{r_{x_i} r + \sigma_i}, \hat{d}_{x_i} = {x_i}^r h^{r_{x_i} r + \sigma_i}\right)$. The decryption of A_i will remain the same: ${x_i}^r$.

Besides the randomness injection, the rest parts of the protocol are the same as the original protocol.

The complete fixed protocol flow is shown in Table 7. For randomness injection, check out steps (ii) and (iii) on the Server side.

Table 7. Security Solution for PSI-CA Protocol 2.

Client, on input $X = \{x_1, \cdots, x_v\}$ $(pk, sk) = (h = g^{\alpha}, \alpha) \leftarrow \mathrm{KGen(par)}$	Public: Server, on input (p, q, g) $Y = \{y_1, \ldots, y_w\}$
(i) chooses $r_{x_1}, \ldots, r_{x_v} \leftarrowtail Z_q$, $E_{pk}(x_i) = (c_{x_i} = g^{r_{x_i}}, d_{x_i} = x_i h^{r_{x_i}})$ $(ii) R_1 = \langle \{E_{pk}(x_1), \ldots, E_{pk}(x_v)\} \rangle$	
$\xrightarrow{R_1}$	
	(i)chooses r $\leftarrowtail Z_q$, computes $\widehat{Y} = \{t_1 = (y_1)^r, \ldots, t_w = (y_w)^r\}$; (ii)for every ciphertext in R_1, chooses $\sigma_i \leftarrowtail Z_q$, $i = 1, \ldots, v$, computes (1)$(E_{pk}(x_i))^r = ((c_{x_i})^r, (d_{x_i})^r)$ (2)$\sigma_{x_i} = (g^{\sigma_i}, h^{\sigma_i})$ (iii)multiply σ_{x_i} with $(E_{pk}(x_i))^r$ to get A_i $A_i = \left(\hat{c}_{x_i} = g^{r_{x_i} r + \sigma_i}, \hat{d}_{x_i} = {x_i}^r h^{r_{x_i} r + \sigma_i}\right)$ (iv) Perm$\{A_1, \ldots, A_v\} =$ $\{(\bar{c}_{x_1}, \bar{d}_{x_1}), \ldots, (\bar{c}_{x_v}, \bar{d}_{x_v})\} =$ $\{\bar{A}_1, \ldots, \bar{A}_v\} = \bar{X}$ (v)$R_2 = \left\langle \widehat{Y} = \{t_1, \ldots, t_w\}, \bar{X} \right\rangle$.
$\xleftarrow{R_2}$	
decrypt $A_i = (\bar{c}_{x_i}, \bar{d}_{x_i})$ for $i = 1, \ldots, v$: $s_i = (\bar{x}_i)^r = \dfrac{\bar{d}_{x_i}}{(\bar{c}_{x_i})^{\alpha}}$ $\lvert X \cap Y \rvert = \lvert \{s_1, \ldots, s_v\} \cap \{t_1, \ldots, t_w\} \rvert$	

Table 8. Second Security Solution for PSI-CA Protocol 2.

Client, on input $X = \{x_1, \cdots, x_v\}$ $(pk, sk) = (h = g^{\alpha}, \alpha) \leftarrow \text{KGen(par)}$	Public : $\{h_1, ..., h_k\}$ $pk = h, g$	Server, on input $Y = \{y_1, \ldots, y_w\}$
(i) Generate IBF_X for X (ii) chooses $r_{c_1}, \ldots, r_{c_m} \leftarrowtail \mathbb{Z}_q$, $E_{pk}\,(IBF_c) = (C_1, ..., C_m)$ where $C_i = \left(g^{r_{c_i}}, g^{IBF_X[i]} h^{r_{c_i}}\right)$ $(iii) R_1 = \{C_1, \ldots, C_m\}$		
	$\xrightarrow{R_1}$	
		(i)On receive $E_{pk}\,(IBF_X) = (C_1, ..., C_m)$: For each y_l where $1 \le l \le w$, evaluate its hashes: $J = \{h_1(y_l), \ldots, h_k(y_l)\}$ (ii)Extract $\{C_{h_1(y_l)}, \ldots, C_{h_k(y_l)}\}$ from R_1. (iii)Multiply all these k ciphertexts: $S_l = \prod_{t=1}^{k} \mathrm{C}_{h_t(y_l)}$ $= \left(g^{\sum_{j \in J} r_{c_j}}, g^{\sum_{j \in J} IBF_X[j]} h^{\sum_{j \in J} r_{c_j}}\right)$ $= Enc_{pk}\left(\sum_{t=1}^{k} IBF_X[h_t(y_l)]\right)$ With random σ_l, where $1 \le l \le w$ We compute $\hat{S}_l = S_l \times (g^{\sigma_l}, h^{\sigma_l})$ $= \left(g^{\sum_{j \in J} r_{c_j} + \sigma_l}, g^{\sum_{j \in J} IBF_X[j]} h^{\sum_{j \in J} r_{c_j} + \sigma_l}\right)$ $\{\bar{S}_l\} = Shuffle\{\hat{S}_l\}$, $R_2 = \{\bar{S}_l\}$
	$\xleftarrow{R_2}$	
Sets $\lvert X \cap Y \rvert = 0$, For $1 \le l \le w$, $\lvert X \cap Y \rvert += 1$ if $Dec(\bar{S}_l) = 0$ Outputs $\lvert X \cap Y \rvert$		

Second Solution Using Bloom Filter and Homomorphic Encryption to construct PSI-CA protocol is common in recent researches [9], [6], [10]. With Homomorphic Encryption, we can perform bit operations on encrypted Bloom Filter.

Since the ElGamal encryption used in PSI-CA protocol 2 is already homomorphic, in this solution, we will try to bring in the Bloom Filter. Moreover, to prevent the Client from performing the exhaustive search attack, we will apply the injection of randomness as well.

Although the original ElGamal encryption is multiplicative homomorphic, to use it on Bloom Filter, it needs modification. The reason lies in that Bloom Filter is of the form $\{0,1\}^m$ and the ElGamal encryption of 0 is of the form: $(g^r, 0)$ in which $r \leftarrow\!\!\!\!- Z_q$. Thereby, an attacker can easily recover a Bloom Filter from an encrypted Bloom filter.

In our construction, we introduce a variant of ElGamal encryption [13]. Its encryption is: $E_{pk}(m) = (\alpha, \beta) = (g^r, g^m h^r)$ in which $r \leftarrow\!\!\!\!- Z_q$ and its decryption is: $\frac{\beta}{\alpha^x} = \frac{g^m (g^x)^r}{(g^r)^x} = g^m$. m can be recovered by running an exhaustive search provided the input of m has low-entropy. The rest parts of this variant are the same as the original ElGamal.

The complete protocol flow of this solution is shown in Table 8.

To sum up, compared to the original PSI-CA protocol 2, we make the following changes in this solution:

- Replace ElGamal encryption with its variant which is additive homomorphic i.e., $E_{pk}(m_1) \times E_{pk}(m_2) = E_{pk}(m_1 + m_2)$.
- Instead of encrypting X, the Client generates the Inverted Bloom filter of X: IBF_X and encrypts it.
- Perform bit operations (bit add) on encrypted Bloom filter and generate S_l based on Server's private input y_l's hash indices: $h_1(y_l), ..., h_k(y_l)$. If y_l is a member of private set intersection, the decryption of S_l should be 0.
- Inject randomness: this part is necessary, otherwise, the Client can still perform the exhaustive search attack on S_l's first element:$g^{\sum_{j \in J} r_{c_j}}$ by using its input: x_i's corresponding $r_{c_{i1}}, \ldots, r_{c_{ik}}$ and recover an invariable: g. As to how to inject randomness, we introduce the following pair:$(g^{\sigma_l}, h^{\sigma_l})$ in which σ_l is random. This could be viewed as an encryption 0, thus, we could homomorphically add this to S_l and it won't affect the decryption result of S_l.

From Table 8, we can see that, compared to the first solution, we made more obvious changes to the original protocol in this solution since we used Bloom Filter.

7 Conclusion

In this paper, we managed to break two PSI-CA protocols in polynomial time.

The vulnerabilities of these protocols come from the invariable introduced by the Server. The Client can recover this invariable from the intermediate results

by an exhaustive search attack. This invariable is further exploited to recover the intersection set. Ergo, by the definition of the PSI-CA, these protocols are broken.

We also ran some attack simulations to prove the viabilities of these attacks. The experiment results also proved the complexity analyses of these attacks.

Furthermore, to solve the security problems of these protocols, we proposed several solutions. Since this paper is mainly about the attack, due to the page limit, we can only cover these solutions' basic constructions and ideas behind these constructions and how they can resist our attacks.

In our following work, we plan to cover more details of these solutions' implementations and formal security proofs. Furthermore, we will try to break more PSI-CA protocols.

References

1. R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 86–97, 2003.
2. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
3. D. Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*, pages 48–63. Springer, 1998.
4. D. Boneh. The decision diffie-hellman problem. In J. P. Buhler, editor, *Algorithmic Number Theory*, pages 48–63, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
5. P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang. On the false-positive rate of bloom filters. *Information Processing Letters*, 108(4):210–213, 2008.
6. A. Davidson and C. Cid. An efficient toolkit for computing private set operations. In *Australasian Conference on Information Security and Privacy*, pages 261–278. Springer, 2017.
7. E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In *International Conference on Cryptology and Network Security*, pages 218–231. Springer, 2012.
8. S. K. Debnath and R. Dutta. Efficient private set intersection cardinality in the presence of malicious adversaries. In M.-H. Au and A. Miyaji, editors, *Provable Security*, pages 326–339, Cham, 2015. Springer International Publishing.
9. S. K. Debnath and R. Dutta. Secure and efficient private set intersection cardinality using bloom filter. In *International Conference on Information Security*, pages 209–226. Springer, 2015.
10. S. K. Debnath, P. Stnic, N. Kundu, and T. Choudhury. Secure and efficient multiparty private set intersection cardinality. *Advances in Mathematics of Communications*, 15(2):365, 2021.
11. C. Dong and G. Loukides. Approximating private set union/intersection cardinality with logarithmic complexity. *IEEE Transactions on Information Forensics and Security*, 12(11):2792–2806, 2017.
12. R. Egert, M. Fischlin, D. Gens, S. Jacob, M. Senker, and J. Tillmanns. Privately computing set-union and set-intersection cardinality via bloom filters. In *Australasian Conference on Information Security and Privacy*, pages 413–430. Springer, 2015.

13. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.
14. S. Hohenberger, A. Sahai, and B. Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 201–220. Springer, 2014.
15. M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 370–389. IEEE, 2020.
16. S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian. Scaling private set intersection to billion-element sets. In *International conference on financial cryptography and data security*, pages 195–215. Springer, 2014.
17. M. Lepinski and S. Kent. Additional diffie-hellman groups for use with ietf standards. Technical report, RFC 5114, January, 2008.
18. Y. Tan and B. Lv. Mistakes of a popular protocol calculating private set intersection and union cardinality and its corrections. *arXiv preprint arXiv:2207.13277*, 2022.
19. E. W. Weisstein. Fermat's little theorem. *https://mathworld. wolfram. com/*, 2004.