A Cryptosystem for Multi-Party Privacy-Preserving Set Union in the Malicious Model

Alison Lin

University of Minnesota

Abstract. There have been several studies on privacy-preserving set operations such as union and intersection, in which the participants can infer the results without learning anything else. Recent papers tend to focus on malicious models. Some of them rely on a trusted third party (TTP); some of them only deal with two parties; others simply assume the existence or a practical implementation of some threshold cryptosystems. This study constructs a new class of homomorphic cryptosystems, called collaborative homomorphic cryptosystem, that can solve the problems practically and directly. Then it shows how to use such a scheme to construct a privacy-preserving protocol. At the end, it also suggests some novel algorithms to increase the scalability of the protocol.

Keywords: Privacy-Preserving, Set Union, Threshold Cryptosystem

1 Introduction

A privacy-preserving set operation scheme was proposed by Kissner in 2005 [6]. It assumes that the players are honest but curious, i.e., they might try to infer other players' data but still follow the protocol. Later papers mostly focus on the problems that arise with malicious users. However, they either assume there are just two parties [3, 5, 8], or rely on a trusted third party (TTP) [5, 7], or assume that an appropriate multi-threshold homomorphic encryption scheme is already given [2], or, assume that an existing non-TTP threshold cryptosystem can be easily applied [4].

All of the studies use homomorphic encryption, since it allows the parties to calculate without knowing the plaintext. This prevents outsiders from knowing the content. However, ideally each player should be able to decrypt only the calculation result but not any of the other individuals' ciphertexts. Therefore, how these parties share the public key and the secret key of the homomorphic cryptosystem is a key problem.

In Kissner's protocol [6], both public key and private key are simply shared by all of the players. This means that there is no encrypted data in the players' point of view. The resistance against the insiders relies on the difficulty of polynomial factorization. Research findings in [3, 5, 8] proposed set operation protocols for two malicious parties. In [5], none of the two players have the secret key - they

can join together to request TTP to decrypt the encrypted calculation. Hazay and Nissim [3] assumes that one of the two parties has the secret key while the other doesn't. The protocol proposed by Lin, Fang, and Cao [7] also requires TTP to initialize for the players. The study [2] deals with a malicious multi-party model based on the assumption that there already exists an appropriate threshold homomorphic encryption scheme E_{pk}, which can be decrypted only when all the players get together. The study [4] claims that its privacy-preserving scheme can be easily converted into a non-TTP version if Pedersen's threshold ElGamal cryptosystem is applied [9].

In this study, we propose a privacy-preserving set union operation protocol in malicious model with at least three parties without a TTP's initializing or decrypting. In our scheme, the players hold the secret key of a homomorphic encryption, but the shared secret key can only decrypt the final calculation result instead of every individual player's encrypted data. The players encrypt the data with their own keys. However these ciphertexts can be combined into the union of their data encrypted by the shared homomorphic encryption scheme. Therefore every player can decrypt the ciphertext of the product.

The scenario for the n-player malicious model is to generate a system of encryption schemes including E_1,...,E_n and E, where the key of each E_i is kept by the player P_i. E is a public-key homomorphic cryptosystem whose private key is shared by all the players. The above keys satisfy $E_1(m_1)\cdots E_n(m_n) = E(m_1\cdots m_n)$, for all $m_1, ..., m_n$. Thus, each player can post their encrypted data $E_i(m_i)$, and calculate $E(m_1\cdots m_n)$ by simply multiplying or adding the ciphertexts together, and then decrypt it by the shared secret key of E to get $m_1\cdots m_n$ without learning each m_i. We call such a system a collaborative homomorphic cryptosystem.

It's not exactly a homomorphism but based on the same idea - computing the ciphertext without decryption. We show that such a system can be generated by a homomorphic encryption (not necessarily fully homomorphic) along with a random number distributing algorithm used to assign each player a secret random number such that the sum or product of these secret numbers is known. We propose such an algorithm without using TTP.

In this study, elements are represented by primes and sets are represented by products of primes. At the end, we discuss the scalability problems and construct two novel algorithms to map finitely many primes into any arbitrarily large set to make the system scalable.

1.1 Our Contributions

The contributions of this paper are a number of novel algorithms including the collaborative homomorphic encryption, several non-TTP secure multi-party computation algorithms such as the private random number distributing algorithm and the player shuffling protocol, the privacy-preserving set union protocol without TTP, two scalable representation algorithms for identifying arbitrarily many elements with finite number of primes, and a secure bulletin algorithm. All the algorithms used in this paper, except the primitive root test and the Chinese

Remainder Theorem, are newly created. They are designed in a straightforward way in order to solve the problems.

1.2 Organization of the Paper

In Section 2, we give the related background. In Section 3, we define our malicious models. In Section 4, we discuss the threshold cryptosystem that is mentioned in [2] and [4], and its limitation. Then we introduce and construct the collaborative homomorphic cryptosystem by modifying a homomorphic cryptosystem. In Section 4, we propose the private random number distributing algorithm used to generate the keys of the collaborative homomorphic encryption. In Section 5, we combine all the above algorithms together to give the privacy-preserving set union protocol without TTP. In Section 6, we suggest three ways to reduce the computation load and to make the system scalable.

2 Background

2.1 Primitive Root

As is well-known, a primitive root modulo N is a multiplicative generator of $\mathbb{Z}_N^\times$. Since the order of any element in $\mathbb{Z}_N^\times$ divides $\phi(N)$, one can determine if an element α is primitive by checking $\alpha^{\phi(N)} \equiv 1 \pmod{N}$ and $\alpha^{\frac{\phi(N)}{p_j}} \not\equiv 1 \pmod{N}$ for each prime factor p_j of $\phi(N)$. In general, for any large r dividing $\phi(N)$, if $\alpha \in \mathbb{Z}_N^\times$, $\alpha^r \equiv 1 \pmod{N}$, and $\alpha^{\frac{r}{p_j}} \not\equiv 1 \pmod{N}$ for all primes $p_j | r$, then α has order r. Primitive roots are frequently used in cryptography due to their high order - the high order property is necessary to the difficulty of the discrete logarithm problem. Primitive roots of $\mathbb{Z}_N^\times$ exist only for $N = 2, 4, p^k$, or $2p^k$ where p is an odd prime and k is a positive integer.

2.2 Homomorphic Encryption

An encryption scheme E is homomorphic if $E(m_1 * m_2) = E(m_1) \star E(m_2)$ for all plaintexts m_1 and m_2, where $*$ and $\star$ denote the operations in the plaintext and ciphertext spaces respectively. A homomorphism is a mapping that preserves the structure of one or more operations on the domain. Homomorphic encryption allows operating directly on ciphertexts without decrypting them.

An additive (resp. multiplicative) homomorphic cryptosystem allows homomorphic computation on only addition (resp. multiplication) on plaintexts. A cryptosytem supports both homomorphic operations is known as fully homomorphic encryption. We describe two of the well-known additive and multiplicative homomorphic encryption schemes below that will be used in this study.

ElGamal ElGamal cryptosystem is the major cryptosystem that we use in this study. We make use of its multiplicative homomorphic property to calculate the product of the participants' private numbers. We show that any multiplicative homomorphic encryption scheme can be converted into a collaborative homomorphic cryptosystem, a new class of the homomorphic cryptosystem that we will introduce and construct in Section 4. With the use of random components during the process, ElGamal is a semantically secure encryption scheme.

Key generation

Public key: g, G, h, where g is a primitive root of the group G and $h = g^x$
Private key: x

Encryption

1. Choose a random number R.
2. The ciphertext of a plaintext m is (g^R, mh^R).

Decryption

1. Calculate $h^R = (g^R)^x$.
2. Calculate $m = mh^R(h^R)^{-1}$.

Paillier Paillier cryptosystem is an additive homomorphic cryptosystem. In this study, the additive homomorphic property of Paillier cryptosystem can be used together with our collaborative homomorphic encryption to privately calculate the sum of the participants' set sizes.

Key generation

1. Randomly choose two large primes p and q with $\gcd(pq, (p-1)(q-1)) = 1$.
2. Let $N = pq$ and $\lambda = \text{lcm}(p-1, q-1)$.
3. Randomly choose $g \in \mathbb{Z}^{\times}_{N^2}$.
4. Ensure $N \mid |g|$.
5. Let $\mu = (L(g^{\lambda} \mod N^2))^{-1} \mod N$, where the function L is defined by $L(u) = (u-1)N^{-1} \mod N^2$.

Public key: n, g
Private key: λ, μ

Encryption

1. Randomly choose $r \in \mathbb{Z}^{\times}_N$.
2. The ciphertext of a plaintext $m \in \mathbb{Z}_N$ is $E(m) = g^m r^N \pmod{N^2}$.

Decryption

Calculate $m = L(E(m)^{\lambda} \mod N^2) \cdot \mu \mod N$.

The homomorphic property is then

$$E(m_1)E(m_2) = (g^{m_1} r_1^N)(g^{m_2} r_2^N) = g^{m_1+m_2}(r_1 r_2)^N = E(m_1 + m_2).$$

3 Security in Malicious Model

Kissner's protocol [6] does not work with the malicious players because its final encryption result is calculated recursively by all the players in turn. The players

have the chance to replace the data of the preceding players by any desired content without being notified by the victims. By doing this, the malicious players can either partially control the result or learn extra information from other users. In our scheme, the players broadcast or post their data calculated and encrypted independently with their own keys. The malicious users can at most corrupt others' posts but have no idea about how it will influence the result. Also, since the posts are public on the bulletin, any player is able to check anytime if his post is corrupted by someone else.

In this paper, we consider two adversary models:

1. In the first model, the adversary does not blindly corrupt others' posts since it leads to a meaningless result and the corruption can be easily obtained by the victim. In this case, any regular online bulletin is enough.

2. In the second model, the adversary could still corrupt others' posts on an unreliable bulletin. The goal of this adversary is nothing but making trouble even though this will stop him from learning anything. In this case, a secure bulletin algorithm is needed. We discuss the secure bulletin in Section 5.2.

4 Threshold Cryptosystem and Collaborative Homomorphic Cryptosystem

4.1 Threshold Cryptosystem and its Limitation

Privacy-preserving studies tend to use homomorphic encryption and secret sharing schemes. This is natural because the multiple parties need to calculate without knowing the individual contents and determine something together. Frikken [2] assumes the existence of a sharing scheme such that the players can decrypt the ciphertext only when they all get together. Hong et al. [4] suggests applying Pedersen's non-TTP threshold ElGamal [9] to implement the non-TTP protocol. We leave the introduction of Pedersen's threshold ElGamal and its comparison with our scheme in the Appendix A.

A threshold cryptosystem is more like a secret sharing scheme. There is only one encryption scheme used by all the players whose secret key can be obtained only when they get together. The private key is even simply held by all the players in Kissner's study [6]. However, whenever the players recover the secret key, they are not only able to decrypt the final ciphertext, but also able to decrypt any data preciously encrypted by the players. At this moment, the information privacy against curious players does not rely on the cryptosystem, but the difficulty of the factorization of polynomials and the play order of the curious players. The problem is : those polynomials are not randomly chosen, in fact, they are used to express the private data and are given by the players separately. If one or two colluding curious players are at special strategic positions, and manipulate their polynomials carefully, they can get an innocent player's plaintext.

How do the studies aiming at malicious models deal with the two problems (malicious handling and weak order)? Some studies suggested adding zero knowledge proofs of correct polynomial multiplication and the knowledge of plaintext

to prevent the malicious players from maliciously handling polynomials [1, 2]. However, knowing the plaintexts and calculating correctly does not directly stop the malicious players from learning information from some innocent player. Besides, shuffling the players's order seems the only defense against the weak order problem. However, the probability that the special orders are assigned to the curious players is at least $\frac{c}{n}$ for some $c > 0$ which is not low enough.

The third problem is efficiency. In most of the published works, the players have to prove the knowledge of every operation they have done, such as the knowledge of the plaintexts, and the correctness of manipulations. This requires additional time and deployment of the data. Moreover, every player P_i needs to wait until all the previous player $P_1, ..., P_{i-1}$ have finished their calculation instead of calculating independently. This causes the malicious handling problem as explained above, and also decreases the efficiency.

The above addressed the problems caused by applying a threshold encryption to the multi-party privacy-preserving set union problem. On the one hand, we hope the secret key can be shared by all the players to decrypt the final result of the homomorphic calculation, but on the other hand, we don't expect any player's encrypted data being decrypted after the secret key is obtained. In the next subsection, we introduce the collaborative homomorphic cryptosystem which treats the players equally. Therefore, first, it won't have weak order problems. Second, any player's outputs are independent instead of being handled through the other players. Everyone's output is posted on the bulletin or broadcast. Therefore, there's no need to prove the correctness of the calculation.

4.2 Collaborative Homomorphic Cryptosystem

A collaborative homomorphic cryptosystem for n players is a system containing the encryption keys $E_1, ..., E_n$ and a homomorphic cryptosystem E such that

$$E_1(m_1) \cdots E_n(m_n) = E(m_1 \cdots m_n)$$

where E is a public key cryptosystem whose secret key is held by all the players and the secret key of each E_i is held only by the player P_i. Thus, each player P_i can encrypt his data m_i by his encryption key E_i. The players are not able to decrypt the ciphertexts of anyone else. However, they can decrypt the product of their chiphertexts since it is encrypted by the shared key E.

A Naive Version of Collaborative Homomorphic Cryptosystem Such a scheme can be implemented with a set of numbers $r_1, ..., r_n$ whose product is 1, and each of these numbers is always only known to the corresponding player. We suggest an algorithm called the private random number distributing algorithm (Protocol 3) in the next subsection to generate such kind of random numbers. Assume that there is already such a set of numbers. We can then define E_i of

our collaborative homomorphic cryptosystem by $E_i(m) = E(mr_i)$. Thus,

$$\begin{aligned} E_1(m_1) \cdots E_n(m_n) &= E(m_1 r_1) \cdots E(m_n r_n) \\ &= E(m_1 \cdots m_n \cdot r_1 \cdots r_n) \\ &= E(m_1 \cdots m_n) \end{aligned}$$

This version is too simple because the random number tuple $(r_1, ..., r_n)$ can be used only once. Otherwise, if an adversarial player gets a plaintext-ciphertext pair $(m, E(mr_i))$ of the target P_i, he can easily obtain P_i's secret key r_i and decrypt all of this victim's ciphertexts. However, repeatedly running Protocol 3 in order to reset $(r_1, ..., r_n)$ for every session is inefficient due to its flooding communication and the $\mathcal{O}(n^2)$ complexity. A better version is stated below.

Normal Version of Collaborative Homomorphic Cryptosystem In fact, the players just need one set of the secret numbers $s_1, ..., s_n$ with sum 0, and these numbers can be reused without being recovered. Assume that they all agree on a primitive root α of $\mathbb{Z}_N^\times$. Instead of posting $E(m_i r_i)$, each player P_i posts $E_i(m_i) \leftarrow E(m_i \alpha^{s_i})$, where $s_1, .., s_n$ is a set with sum 0. Thus,

$$\begin{aligned} E(m_1 \alpha^{s_1}) \cdots E(m_n \alpha^{s_n}) &= E(m_1 \cdots m_n \cdot \alpha^{s_1} \cdots \alpha^{s_n}) \\ &= E(m_1 \cdots m_n \cdot \alpha^{s_1 + \cdots + s_n}) \\ &= E(m_1 \cdots m_n) \end{aligned}$$

As long as the players keep $s_1, .., s_n$ private, they can update the random numbers $r_i = \alpha^{s_i}$ with product 1 by renewing α offline. The first primitive root should be generated carefully so that the players are all convinced with the randomness and hence we construct it by flooding as shown in Protocol 1. Once the players have the agreement on a generator, they can then use a hash function or a collectively agreed fixed random function to compute other generators (Protocol 2).

Protocol 1: The first primitive root determining algorithm (online)
Input: n players $P_1, ..., P_n$; N
Require: Jointly determine a primitive root α modulo N.
1. **for** each player P_i
   (a) Choose a random number α_i
   (b) Send the α_i to all the other players
2. **for** each player P_i
   (a) Calculate $\alpha \leftarrow \sum \alpha_i$
   (b) Apply the primitive root test described in Section 2.1
      if α is a primitive root
         Return α
      else
         Go to 1.

Remark. The number of primitive roots modulo N is $\phi(\phi(N))$ which ensures enough choices for primitive roots.

Protocol 2: Random primitive roots extension algorithm (offline)
Input: n players $P_1, ..., P_n$; r; N; a jointly determined primitive root α; a hash function or a known random function H
Require: Determine r random primitive roots $\alpha^{(1)}, ..., \alpha^{(r)}$ modulo N.
1. $\alpha^{(0)} \leftarrow \alpha$
2. **for** $k = 1, ..., r$
   (a) $\alpha \leftarrow \alpha^{(k-1)}$
   (b) $\alpha \leftarrow H(\alpha)$
   (c) **if** α is a primitive root
   $\alpha^{(k)} \leftarrow \alpha$
   else
   Go to 2.(b)
3. Return $\alpha^{(1)}, ..., \alpha^{(r)}$

4.3 Private Random Number Distributing Algorithm

In order to construct the collaborative homomorphic encryption scheme by modifying a homomorphic encryption, we need an algorithm to distribute to every player a secret number, such that the sum (resp. product) of these numbers is 0 (resp. 1). This can be done easily if there is a TTP in charge of generating and assigning the numbers to the players separately. For the non-TTP case, we do this by having each player send random numbers to the next ℓ players separately, as shown in Protocol 3. Thus, each player would also receive random numbers from the preceding ℓ players and then calculate his own secret number s_i by adding up all of the received numbers and subtracting all of the numbers he sent out. Thus, the sum of the players's results will be 0 since all the inputs and outputs cancel each other when adding up.

The only concern here is the confidentiality of s_i with respect to the choice of ℓ. If the 2ℓ players consisting of the ℓ before and ℓ after collude, a player's secret would be compromised. To prevent s_i from being inferred by the 2ℓ colluding players, in practice, we choose ℓ to be $\lfloor \frac{n}{2} \rfloor$ which means flooding. In other words, each player's secret number is related to all of the other players. Hence a player's secret number can be compromised only if all the other players are malicious. When choosing ℓ to be $\lfloor \frac{n}{2} \rfloor$, the complexity of Protocol 1 is $\mathcal{O}(n^2)$, but it is applied only once at very beginning of the privacy preserving protocol. Once these random numbers are all set, the players can then use Protocol 2 to efficiently construct a series of primitive roots and reuse those s_i's repeatedly.

Remark. In protocol 3, it is obvious that $\sum s_i = 0$. We leave the proof in Appendix B.

Protocol 3: Private random number distributing algorithm without TTP
Input: n players $P_1, ..., P_n$. Security parameter ℓ
Require: Each player P_i is assigned a random number s_i with $\sum s_i = 0$

1. **for** each player P_i
   (a) Choose ℓ random numbers $a_{i1}, ..., a_{i\ell}$
   (b) Send the numbers to $P_{i+1}, ..., P_{i+\ell}$ separately, i.e., send a_{ij} to P_{i+j}, for $j = 1, ..., \ell$
   (c) Receive $a_{i-t,t}$ from P_{i-t}, for $t = 1, ..., \ell$
   (d) Calculate $s_i \longleftarrow \sum_{t=1}^{\ell} a_{i-t,t} - \sum_{j=1}^{\ell} a_{ij}$

Remark. The scheme can be converted into the multiplicative version. By fixing a multiplicative generator g, $s_1 + \cdots + s_n = 0$ induces $\prod g^{s_i} = 1$, and hence this generates n random numbers whose product is 1.

Lemma 1. *Suppose that the ratio of malicious players is γ, where $0 \leq \gamma < 1$. Let P_i be one of the n players with secret number s_i. The probability that s_i is inferred by other players is bounded by $\gamma^{2\ell}$.*

Proof. See Appendix C.

4.4 Players Shuffle Algorithm

In practice, if there are n players, we choose ℓ to be $\lfloor \frac{n}{2} \rfloor$ in Protocol 3. Thus, each player's secret random number is generated by all of the other players together, and hence can be compromised only when all the other players collude. Therefore the malicious players can not take advantage from the order of the players. However, if for some reason, we need choose ℓ to be a smaller number, then a player's secret random number is generated by 2ℓ of certain players. In order to minimize the advantage of the adversaries from technically choosing order, we need to randomize their positions. We suggest a simple multi-party computing algorithm (Protocol 4, see Appendix E.) for determining a random permutation of $[n]$.

5 Privacy-Preserving Set Union Protocol in Malicious Model

We identify each element of the set with a prime number less than N. Thus, a product of primes represents the set of the corresponding elements identified with those primes. By the unique factorization characteristic of the integers, the union of sets corresponds to the product of the numbers representing the sets. Thus, the set union problem can be identified with the number multiplication problem. We state the non-TTP privacy-preserving number multiplication protocol below (Protocol 5), and deal with the scalability problems in the next section.

Protocol 5: Privacy-preserving multiplication without TTP
Input: n players $P_1, ..., P_n$, each has r secret numbers $m_i^{(1)}, ..., m_i^{(r)}$; a jointly determined primitive root α; a multiplicative homomorphic encryption scheme E with secret keys shared by all the players
Require: Return $\prod_i m_i^{(k)}$ to all the players for $k = 1, .., r$ without leaking any $m_i^{(k)}$ to P_j where $i \neq j$

1. The players jointly randomize the order of the players (Protocol 4)
2. The players use Protocol 3 to determine s_1,...,s_n, where $\sum s_i = 0$ and s_i is only known to P_i
3. The players use Protocol 2 to determine primitive roots $\alpha^{(1)}, ..., \alpha^{(r)}$
4. **for** $k = 1, ..., r$
   for each player P_i
   $\alpha \leftarrow \alpha^{(k)}$
   Post $E(m_i^{(k)}\alpha^{s_i})$ on the bulletin
   for each player P_i
   (a) Calculate $E(\prod_i m_i^{(k)}) \leftarrow \prod_i E(m_i^{(k)}\alpha^{s_i})$
   (b) Decrypt $E(\prod_i m_i^{(k)})$ to get $\prod_i m_i^{(k)}$

5.1 Security against the Malicious Players

Semantic Security Since the secret key of E is held by all the players, any player can decrypt a player P_i's data $E(m_i\alpha^{s_i})$ to discover $m_i\alpha^{s_i}$. If a malicious player sends the challenger two plaintexts m and m' and the challenger returns either $m\alpha^{s_i}$ or $m'\alpha^{s_i}$. If $m = \alpha^a$ and $m' = \alpha^b$, the adversary needs to distinguish if the result is α^{s_i+a} or α^{s_i+b}. Since s_i is unknown to the adversary, $s_i + a$ and $s_i + b$ are indistinguishable. Therefore the scheme is semantically secure.

Known-Plaintext Security In this part we show that the KPA security of our scheme against insiders is equivalent to the KPA security of ElGamal cryptosystem. Suppose that an adversarial player accidentally obtains the plaintext of $m\alpha^{s_i}$ of the player P_i at some round and then hence recovers α^{s_i}. In order to use this information to exploit another ciphertext, say, $m'\alpha'^{s_i}$ in another round, the adversary needs the knowledge of the relation between α and α'. For example, if he knows $\alpha' = \alpha^2$ then α'^{s_i} is $(\alpha^{s_i})^2$. However, obtaining the relation between α and α' is equivalent to breaking the ElGamal cryptosystem. Indeed, the ciphertexts of m and m' encrypted by Elgamal are (g^R, mh^R) and $(g^{R'}, m'h^{R'})$, where $h = g^x$ and x is the secret key. Since g is a high order element and R, R' are random, we can view g^R as α and $g^{R'}$ as α' in our case, and hence these two ciphertexts have the form $(\alpha, m\alpha^x)$ and $(\alpha', m'\alpha'^x)$ where x is unknown to the adversary, similar to the s_i in our scheme. Therefore, obtaining the relation between α and α' is equivalent to obtaining the relation between g^R and $g^{R'}$.

5.2 Secure Bulletin Algorithm

In our first malicious model, we assume that the adversary doesn't modify others' ciphertexts since he can not learn or control any part of the results by doing this. In this model, any broadcasting system or regular online bulletin would work well enough. In the second malicious model, we assume that the attacker could still try to corrupt the data on the bulletin posted by other players even though the corruption could be obtained when the victim simply checks the bulletin. The secure bulletin is built for this case. It is supposed to satisfy

- integrity - the ciphertext posted by an user can not be modified by others without being detected;
- undeniability - a player can not deny his posts;
- verifiability - any players can verify if a post is indeed posted by a certain player.

A secure bulletin can be implemented through either algorithms or technical system designs. For example, we can simply have each player manage his own broadcasting system. The secure algorithm is needed only in the worst case - all of the results have to be posted on an unreliable public space. To achieve such an algorithm, we can also have the players sign their data by separate signature keys, or use a time-released cryptosystem to ensure that they are not able to get any information during a certain period of time. Here we attempt to propose a secure bulletin algorithm by using the secret numbers s_i's.

Protocol 6: Secure bulletin algorithm
Input: n players where each player P_i has data D_i to post; a primitive root β; a public online space X; a hash function H
Require: The players are convinced with the source of the data.

1. **for** each player P_i
   Post $H(\beta^{s_i}||D_i)$ on X
2. **for** each player P_i
   (a) Collect the n posts from X
   (b) Post (β^{s_i}, D_i) on X
3. **for** each player P_i
   (a) Collect the n posts (b_j, d_j) from X
   (b) Accept the data $D_1 \leftarrow d_1, ..., D_n \leftarrow d_n$, if
      (i) $D_j = d_j$ for all $j = 1, ..., n$
      (ii)$\prod b_j = 1$ $\ll$*product test*$\gg$
      (iii) $H(b_j||d_j) = H(\beta^{s_j}||D_j)$ for all $j = 1, ..., n$ $\ll$*hash test*$\gg$

Lemma 2. *Protocol 6 satisfies the integrity, undeniability, and verifiability illustrated in above.*

Proof. See Appendix D.

6 Reduce the Computation Load and Make the Representation Scalable

6.1 Parameter Setting

Let S denote the set that we want to describe, K denote the number of primes we use to represent the set elements, $U \subseteq S$ denote the union of the players' sets, and N be the modulus of the ElGamal cryptosystem. In order to have enough primes to identify with S, K should be at least as large as $|S|$. Assume that the K prime numbers are $p_1, ..., p_K$ in the increasing order. In case that the product of the primes corresponding to U exceeds N, the largest possible product, $p_K^{|U|}$, should be bounded by N. In conclusion, we have the following two conditions.

1. $|S| \leq K < N$
2. $p_K^{|U|} \leq N$

To obtain $|U|$ without TTP, the players can post the sizes of their subsets encrypted by an additive collaborative homomorphic encryption scheme made by Paillier cryptosystem satisfying $E_1(m_1)+\cdots+E_n(m_n) = E(m_1+\cdots+m_n)$. Thus, the players can calculate $|U|$ by summing up the ciphertexts and decrypting it. Certainly, if the players don't mind releasing the number of their sets, they can simply publish those.

The two conditions above give rise to several problems. First, although N is usually chosen to be a great number with thousands of bits, if $|S|$ is very large, then in order to have enough primes, N has to be extremely large but this would seriously pull down the efficiency. Second, by Condition 2, we see that the gap between p_K and N could be exponentially large along with the increase of $|U|$. Thus the number of the usable primes would be very low if we don't allow N to be an astronomical number which is, however, impossible to deal with.

In the rest subsections, we suggest three ways to solve the problems which break the two conditions above and allow the choice of K to be independent from $|S|$. Any of the three methods can be combined together to improve the scalability and efficiency. In Section 6.2, we propose an idea to split the computation and representation into smaller pieces by the Chinese Remainder Theorem. In Section 6.3, we show that S can also be split into small pieces to be calculated separately. This allows us to reuse a fixed set of primes $p_1, ..., p_K$ to represent all the elements in S. In Section 6.4, we deal with the case where $|U| \ll |S|$. Our algorithm allows K to be as small as $|U|$ instead of being as large as $|S|$.

6.2 Split the Number into Small Pieces

Note that in our setting, N has the form either p^k or $2p^k$. In this case the Chinese Remainder Theorem doesn't meaningfully reduce the computation load since the coprime factorization of N is trivial. One can modify the scheme in order to apply the Chinese Remainder Theorem. For example, choose N to be a

large number with nontrivial coprime factorization, and use the primitive roots of a large order subgroup of $\mathbb{Z}_N^\times$ instead of the primitive roots of $\mathbb{Z}_N^\times$.

To reduce the load of multiplicative and exponential computation modulo N, we can use the Chinese Remainder Theorem to cut N into small pieces. Factor N into several mutually coprime numbers $q_1, ..., q_t$. Then $\mathbb{Z}_N = \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_t}$. Thus, by splitting the numbers of $\mathbb{Z}_N$ into t small numbers, the exponential and multiplicative operation load can be reduced.

6.3 Scalabale Representation - Dynamic Entries Mapping

If $|S|$ does not have a potential limit, it won't be a good idea to increase the number of primes K to represent the entire S by increasing N without limitation, as it devastates the computation efficiency. In fact, we can use a certain amount of primes to represent infinitely many data. To represent the elements in S, we can simply cut S into a number of subsets $S_1, ..., S_L$, with each $|S_j| \leq K$. Use the primes $p_1, ..., p_K$ to represent the items in each S_j. Then each player P_i simply post an L-tuple or $(j, E(m_i^{(j)} \alpha_j^{s_i}))$ for $j = 1, ..., L$ to indicate the subsets. After the algorithm returns the union in each S_j separately, the players can simply get the total set union by combining these sub-unions together.

In practice, we can filter out those "redundant" S_j's in advance. At beginning, the players use an additive collaborative homomorphic cryptosystem to calculate $|U \cap S_j|$ for $j = 1, ..., n$, and then drop those S_j's with $|U \cap S_j| = 0$, i.e., contributing nothing to the union U.

6.4 Scalabale Representation - Dynamic Pairs Mapping

Consider the case when S is extremely large, and the number of the set union is much less than $|S|$, i.e., $|U| \ll |S|$. If we divide S into L subsets with $|S_j| \leq K$ as in 6.3, L could be extremely large, meaning that we need to produce a lot of primitive roots to deal with each S_j. If we restrict the number of the subsets, L, then each S_j could be still very large while $|U \cap S_j|$ is still much smaller than $|S_j|$. In this subsection, we propose a way to balance the choice of L and $|S_j|$, which also gives rise to the optimization problem of the combination of the two methods. Restricting L results in $|U \cap S_j| \ll |S_j|$. $|U \cap S_j|$ can be reduced along with the increase of L. Such a subproblem brings us back to the original problem, and hence in what follows, we go back to the original case.

Consider the case when $|U| \ll |S|$ where $|U|$ is small enough. The idea is to map K primes to the entries in U instead of in S as in previous sections. To do this, we identify S with a cross product of linearly ordered sets, say, $S = A \times \cdots \times A = A^t$. Choose K to be a number larger than or equal to $|U||A|$.

For any positive integer m, let $[m]$ denote the set $\{1, ..., m\}$. For $x \in S = A^t$ and $I \subseteq [t]$, define x_I to be the coordinates of x indexed by I, and define $B_I := \{x_I \mid x \in B\}$ for any subset $B \subseteq S = A^t$. For example, if $x = (x_1, ..., x_t) \in S = A^t$ then $x_{[3]} = (x_1, x_2, x_3)$.

In this algorithm, primes are used to represent pairs instead of entries. The players look at the posted tuples in each of the $t-1$ rounds and post the product

of primes representing the extended tuples they have. The primes representation are dynamically updated in different rounds. In the first round, the players only look at the first two coordinates of their elements. They post the primes corresponding to the pairs they have and apply Protocol 5 (the privacy preserving multiplication protocol) to obtain the union of these pairs. We call the set $currU$ (current union). Then the players identify the pairs in $currU \times A$ with the primes and apply Protocol 5 to obtain the union again. In each round, the players obtain the union of a longer tuples, until the tuples have length t.

Protocol 7: Partial union constructing algorithm
Input: n players $P_1, ..., P_n$; primes $p_1, ..., p_K$, where $K \geq |U||A|$;
$S = A^t = A \times \cdots \times A$; a lexically ordered set $currU$
Return: U, the union of all the players' subsets of $S = A^t$

1. $currU \leftarrow A$
2. **for** $i = 2, ..., t$
   (a) The players remove the repeated items from $currU$
   (b) The players lexically order $currU$
   (c) The players identify the elements of $currU \times A$ with the first $|currU||A|$ primes of $p_1, ..., p_K$ in order
   (d) The players use Protocol 5 to obtain $U_{[i]}$
   (e) $currU \leftarrow U_{[i]}$
3. $U \leftarrow currU$

7 Conclusions

We converted the privacy-preserving set union problem into the privacy-preserving multiplication problem by identifying set entries with primes. Under this identification, we proposed multi-party privacy-preserving set union without TTP. To do this, we introduced the collaborative homomorphic cryptosystem and suggest a private distributing algorithm to construct it. At the end, we provided three ways to increase the scalability of the representation.

References

1. R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In Proceedings of Advances in Cryptology - EUROCRYPT01, volume 2045 of Lecture Notes in Computer Science, pages 280-299, 2001.
2. K. Frikken. Privacy-Preserving Set Union. Proceedings of the 5th international conference on Applied Cryptography and Network Security, June 05-08, 2007.
3. C. Hazay, K. Nissim. Efficient set operations in the presence of malicious adversaries. In Public Key Cryptography, LNCS, pages 312-331. Springer, 2010.
4. J. Hong, J. W. Kim, J. Kim, K. Park, and J. H. Cheon. Constant-Round Privacy Preserving Multiset Union. Cryptology ePrint Archive, Report 2011/138, 2011. http://eprint.iacr.org/.
5. M. Kantarcioglu, O. Kardes. Privacy-preserving data mining in malicious model. Technical Report CS-2006-06, Stevens Institute of Technology, 2006.
6. L. Kissner, D. Song. Privacy-preserving set operations. In Advances in Cryptology, CRYPTO05, volume 3621 of LNCS, pages 241-257, 2005.
7. H. Lin, Y. Fang, Z. Cao. Private Information Extraction over Online Social Networks. Cryptology ePrint Archive, Report 2011/446, 2011, http://eprint.iacr.org/2011/446.pdf
8. Y. Lindell , B. Pinkas. Privacy Preserving Data Mining. Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology, pages 36-54, August 20-24, 2000
9. T. P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In EUROCRYPT, pages 522-526, 1991.
10. Y. Sang and H. Shen. Efficient and secure protocols for privacy-preserving set operations. In ACM Transactions on Information and Systems Security, 2009

Appendix

A. Pedersen's Threshold ElGamal

Pedersen's Threshold ElGamal is a non-TTP private key sharing scheme using ElGamal encryption. Our non-TTP collaborative homomorphic cryptosystem is also obtained by using homomorphic encryption. In order to compare our scheme with threshold cryptosystem, we first study Pedersen's threshold cryptosystem [9] below.

Pedersen's Threshold ElGamal

Let g be the public generator of an ElGamal cryptosystem. Each player posts $h_i = g^{x_i}$, where x_i is only known to the ith player. The public key h of the ElGamal encryption is calculated by $\prod h_i$. Thus, the secret key of this ElGamal cryptosystem is $x = \sum x_i$. Thus, when all the players get together, they can decrypt everything by sharing their x_i's together. The Pedersen scheme uses (k, n)-threshold secret sharing of x. It generalized Shamir's secret sharing scheme to share the secret key x, so that it can be recovered when any k members get together. For $k = 1$ or $k = n$, it is trivial since it can be implemented by simply assigning x to every participant or having all the players share their x_i's respectively.

Pedersen's scheme suggests a good way to share the secret key of the ElGamal cryptosystem, which is the only cryptosystem used by the members. It doesn't fit our needs well. On the one hand, we hope the secret key can be shared by all the players to decrypt the final result of the homomorphic calculation, but on the other hand, we don't want to see each player's encrypted data being decrypted after the secret key is obtained. So if we apply Pedersen's scheme to our privacy-preserving set union problems, we would need to make quite a few extra modification to it. In what follows, we introduce the concept of the collaborative homomorphic cryptosystem which allows the participants to keep their individual privacy even though they share the secret key of the ElGamal cryptosystem. We modify Pedersen's threshold ElGamal into a collaborative homomorphic cryptosystem. Finally, we introduce our collaborative homomorphic cryptosystem which is designed directly for the privacy-preserving problems.

Can Pedersen's Threshold ElGamal be Converted into a Collaborative Homomorphic Cryptosystem?

In order to see how Pedersen's threshold ElGamal works in the privacy-preserving protocol, here we make attempt to convert Pedersen's scheme into a collaborative homomorphic encryption. Recall that the player P_i has $h_i = g^{x_i}$ and $h = \prod h_i = g^x$ is the public key of the ElGamal encryption, where each x_i is only known to P_i but x is known to all the members. We can define $E_i(m_i) = (g^R, m_i h_i^R)$, where R is a number public to all the participants but

not outsiders. Therefore, once the players get all of the $E_i(m_i)$'s, they can then multiply the second coordinates together to obtain $\prod(m_i h_i^R) = h^R \prod m_i$. Thus, they can obtain $\prod m_i$ by multiplying h^{-R} on it. In the original ElGamal encryption, R is chosen randomly by the encryptor, but this R has to be consistent among all the players here in order to keep the homomorphic relation. However, this scheme is insecure since if R and all the h_i's are public to all the players, then any player can obtain m_i by multiplying h_i^{-R} to $m_i h_i^R$, even though he does not have x_i. Besides, another problem in this scheme would be how can P_i still keep x_i private even after x is obtained or shared by all the members?

B. A proof for Protocol 3

Lemma. *With $s_1, ..., s_n$ given by Protocol 3, $\sum s_i = 0$.*

Proof.

$$\begin{aligned}
\sum_{i=1}^{n} s_i &= \sum_{i=1}^{n} \left[\sum_{k=1}^{\ell} \alpha_{i-k,k} - \sum_{j=1}^{\ell} \alpha_{ij} \right] \\
&= \sum_{k=1}^{\ell} \sum_{i=1}^{n} \alpha_{i-k,k} - \sum_{j=1}^{\ell} \sum_{i=1}^{n} \alpha_{ij} \\
&= \sum_{k=1}^{\ell} \sum_{t=1}^{n} \alpha_{tk} - \sum_{j=1}^{\ell} \sum_{i=1}^{n} \alpha_{ij} \\
&= 0
\end{aligned}$$

C. A proof for Lemma 1

Lemma. *Suppose that the ratio of malicious players is γ, where $0 \leq \gamma < 1$. Let P_i be one of the n players with secret number s_i. The probability that s_i is inferred by other players is bounded by $\gamma^{2\ell}$.*

Proof. The number of malicious players is γn. After randomizing the order of the players by Protocol 4, the probability that a player is malicious is $\frac{\gamma n}{n}$. Since s_i is determined by another 2ℓ players, in order to compromise s_i, the adversary needs to collude with these 2ℓ players. The probability that the 2ℓ players are all malicious is given by

$$\begin{aligned}
\frac{\binom{\gamma n}{2\ell}}{\binom{n}{2\ell}} &= \frac{\gamma n}{n} \cdot \frac{\gamma n - 1}{n - 1} \cdot \frac{\gamma n - 2}{n - 2} \cdots \frac{\gamma n - (2\ell - 1)}{n - (2\ell - 1)} \\
&< \underbrace{\gamma \cdots \gamma}_{\ell} \\
&= \gamma^{2\ell}.
\end{aligned}$$

D. A proof for Protocol 6

Lemma. *Protocol 6 satisfies the integrity, undeniability, and verifiability illustrated in above.*

Proof. The capability of a player to pass the product test with other players ensures the verifiability and undeniability. For the integrity, if an adversary P_j replaces P_i's second post (β^{s_i}, D_i) by (β^{s_i}, D_i'), then he would also need to replace P_i's first post by $H(\beta^{s_i}||D_i')$ in order to pass the product test on Step 3. However, he can predict neither β^{s_i} nor $H(\beta^{s_i}||D_i')$ until P_i's second post, while the first posts have already been recorded by all the players. It is also meaningless for the adversary to tamper a player P_i's post in the first round, when he has no idea about β^{s_i}. The only way to pass both of the product test and the hash test with forged data, is to tamper all of the posts in both rounds. However, any honest players will not accept it if the fraudulent data are not identical to their posts.

E. Players shuffle algorithm (Protocol 4)

Protocol 4: Players shuffle algorithm
Input: n players $P_1, ..., P_n$.
Require: Jointly determine a random permutation $(x_1, ..., x_n)$ of the set $\{1, ..., n\}$. All the players should be convinced with the randomness of the output.

1. **for** each player P_i
   (a) Generate an n-tuple $(x_{i1}, ..., x_{in})$, where each x_{ij} is chosen from $\{1, ..., n\}$ randomly
   (b) Post the n-tuple on the bulletin
2. $A \leftarrow \{1, ..., n\}$
3. **for** $i = 1, ..., n$
   (a) $t \leftarrow \sum_{j=1}^{n} x_{ij} \pmod{n - i + 1}$
   (b) $x_i \leftarrow$ the t-th entry of A
   (c) $A \leftarrow A \setminus \{x_i\}$
4. Use the permutation represented by $(x_1, ..., x_n)$ to reorder the players $P_1, ..., P_n$