Privacy-Preserving Set Union

Keith Frikken

Department of Computer Science and Systems Analysis
Miami University
Oxford, OH 45056

Abstract. Recently there has been a significant amount of work on privacy-preserving set operations, including: set intersection [14,6,21,9], testing set disjointness [17], multi-set operations [18], and set union [16,1,18]. In this paper, we introduce novel protocols for privacy-preserving set union in the malicious adversary model. More specifically, each participant inputs a set of values, and at the end of the protocol, each participant learns the items that are in at least one participant's set without learning the frequency of the items or which participant(s) contributed specific items. To our knowledge our protocol is the most efficient privacy-preserving set union protocol for the malicious adversary model to date.

1 Introduction

Recently there has been a significant amount of work on privacy-preserving set operations, including: set intersection [14,6,21,9], testing set disjointness [17], multi-set operations [18], and set union [16,1,18]. In this paper, we introduce a new protocol for privacy-preserving set union ($\mathcal{PPSU}$) in the malicious adversary model. We are only aware of one other $\mathcal{PPSU}$ protocol that is secure in the malicious adversary model, which was introduced in [18], and our new protocol is more efficient than this protocol. An application of this work is the following scenario: Suppose several hospitals treat a rare disease and that a research group needs various information about patients with the disease in order to develop alternative treatments. Because cases of the disease are so sparse, the research group needs more than a single hospital's data. Unfortunately, some patients may have gone to multiple hospitals and these patients' information will taint the quality of the collected data. By using a secure set union protocol the research group can gather the information from multiple hospitals while omitting duplicate patients without learning the identity of the patients. Furthermore, $\mathcal{PPSU}$ has been used as a building block in some privacy-preserving data mining protocols [16,23] and privacy-preserving graph algorithm protocols [1], and thus we believe that improved protocols for $\mathcal{PPSU}$ will be useful in many application domains.

Contributions: The contributions of this work are not only in a preliminary protocol for $\mathcal{PPSU}$, but also in several extensions of this protocol. We now summarize our results:

J. Katz and M. Yung (Eds.): ACNS 2007, LNCS 4521, pp. 237–252, 2007.

- *Preliminary Protocols:* Our preliminary protocols securely compute the set union for: i) two parties in the honest but curious adversary model with $O(n)$ communication, ii) multiple parties in the honest-but-curious adversary model with $O(k^2n)$ communication, and iii) multiple parties in the malicious adversary model with $O(n^2k^2 + k^3n)$ communication (for sets of size n and k participants). This is described in section 5. Note that for the complexities given in this paper there is always an implicit security parameter.
- *Padding:* Our preliminary protocols reveal the number of items in each participant's sets. In section 6.1 we introduce modifications to our preliminary protocol that allow participants to pad their sets with "dummy" items in order to obfuscate this information. This is described in section 6.1.
- *Cardinality:* For some applications computing the size of the set union without revealing the actual union is preferred. In section 6.2 we introduce a protocol that reveals only the cardinality of the set union.
- *Empty-set attack:* One problem with any set union protocol is that a dishonest participant can use the empty-set as their input set. This is particularly damaging if there are few participants; e.g., when there are only two participants then this reveals the honest party's set. In section 6.3 we introduce counter-measures against this attack.
- *Over-threshold set union:* As discussed in [18] there are some situations where participants want to know all items that are in at least t sets. In section 6.4 we introduce protocols that compute this "over-threshold" set union.

Outline: The rest of this paper is organized as follows. In section 2 we formally describe the set union problem and the security models considered in this paper. In section 3 we provide a detailed summary of previous work in privacy-preserving set operations. In section 4, we introduce several building blocks that are used in our protocols, but which are not contributions of this paper. In section 5 we introduce preliminary protocols for privacy-preserving set union. In section 6, we describe several extensions to the preliminary protocols. Finally, we conclude our paper in section 7.

2 Problem Definition

There are k participants, labeled $P_1, \ldots, P_k$, that have respective sets $S_1, \ldots, S_k$ that are drawn from a universe of items $\mathcal{U}$. As a shorthand notation, we use η_i to represent $|S_i|$. To denote the specific items in a set S_i we use the notation $(S_i)_1, \ldots (S_i)_{\eta_i}$. Note that the above notation implies an ordering on the items, but this is just for ease of notation; that is, we do not assume the items are in any specific order. To simplify the notation when giving the complexity analysis of our protocols we use $n = \max_{i=1}^{n} \eta_i$ and we assume that every party has n items. The desired output of the protocol is that the participants learn $\cup_{i=1}^{k} S_i$.

We define security in the standard way (see [10] for more details), that is we define an ideal model (using a trusted third party) and show that any polynomial-time adversary in our protocol can be simulated by a polynomial-time adversary in the ideal model.

Ideal Model: In the ideal model the participants send their sets to a trusted third party T, and then this party broadcasts the union of the sets along with the size of each individual set to all of the participants. Note that this reveals slightly more than the set union, specifically each participant's set size is revealed.

Honest-But-Curious Adversaries: In this adversary model the participants will faithfully follow the prescribed protocol, but will try to learn additional information after the protocol. To prove security in this model, we show that the transcript that is produced by our protocol could be simulated by an adversary that has the output of the protocol. More specifically, we require that the simulator be able to generate a transcript that is computationally indistinguishable from the real transcript.

Malicious Adversaries: In the malicious adversary model the adversary will deviate from the protocol in an arbitrary fashion. The purpose of this deviation can be several things, including: i) to learn more information about honest participants' values, ii) to change the result of the protocol, or iii) to terminate the protocol prematurely. In this paper we do not consider early termination to be a problem (although our protocols could be modified to prevent this using standard techniques such as [11]). Thus to show our protocol is secure in this model we show that: i) any transcript generated by the protocol can be simulated given the results of the protocol and ii) that any result-changing action by an adversary could be achieved by changing the adversary's inputs in the ideal protocol.

3 Related Work

The $\mathcal{PPSU}$ problem can be solved with the generic results of secure multiparty computation [24,11]. While recent advances in malicious circuit evaluation [3] show that it is possible to simulate a circuit efficiently in the malicious model, the communication complexity of such a the scheme will still be the number of gates in the circuit times a security parameter times a polynomial of k (the specific polynomial depends on the scheme being used). The straight-forward circuit for set union has $O(k^2n^2 \log |\mathcal{U}|)$ gates (when given k parties whose sets each contain n elements).

As mentioned earlier, many privacy-preserving protocols have been introduced for set operations other than set union, including: set intersection [14,6,21,9], testing set disjointness [17], and multi-set operations [18]. One might think that, because of DeMorgan's Law, a secure protocol for set union follows directly from a secure protocol for set intersection. Specifically, one can compute $\cup_{i=1}^{n} S_i$, by computing $\overline{\cap_{i=1}^{n} \overline{S_i}} = \mathcal{U} - (\cap_{i=1}^{n} (\mathcal{U} - S_i))$. While this method does correctly compute the set union, when $\mathcal{U}$ is significantly larger than $\cup_{i=1}^{n} S_i$ this method is inefficient. Thus, the results of this paper are most beneficial for applications where the sets are chosen from large domains.

There have also been several protocols that privately compute set union [16,1,18]. However, the previous solutions have not been fully satisfactory

solutions. In [16], the protocol reveals superfluous information, such as the cardinality of the intersection between some participants' sets, in order to improve efficiency. The protocols in [1] was proven secure only in the honest-but-curious adversary model (this protocol was for two parties and required $O(n \log |U|)$ communication). Finally, the protocol given in [18] is secure in the malicious model, but according to our analysis[1] the communication complexity of their honest-but-curious approach is $O(k^3 n^2)$, whereas our scheme has communication complexity $O(k^2 n^2 + k^3 n)$.

4 Building Blocks

In this section we outline the building blocks that are used by our protocols.

4.1 Homomorphic Encryption

In this paper we use a public-key semantically-secure [12] additively homomorphic encryption scheme, such as [22]. Throughout this paper we will denote the encryption and decryption functions by E_{pk} and D_{sk} respectively. Recall that it is possible to add the plaintexts of two encrypted values by multiplying the ciphertexts; that is, when given the encryptions $E_{pk}(x)$ and $E_{pk}(y)$, we can compute $E_{pk}(x+y)$ by computing $E_{pk}(x) * E_{pk}(y)$. Also, when given $E_{pk}(x)$ it is possible to compute $E_{pk}(c*x)$ for any constant c by computing $E_{pk}(x)^c$. Finally, we use homomorphic schemes where it is possible to re-encrypt a ciphertext value to generate another ciphertext with the same plaintext value.

We utilize a threshold version of Paillier's scheme throughout this paper, such as the one presented in [2,4,8]. More specifically, we require a (k,k)-threshold decryption algorithm, that is, the decryption key is distributed among all k players, and the participation of all k players are required to decrypt a value. We use the same model as [18], and we assume that the threshold keys have already been distributed amongst the participants. The communication required to perform a joint decryption is $O(k)$.

4.2 Polynomial Representation of Sets

Several previous set operation results use polynomials to represent sets or multi-sets [9,18]. Specifically, to represent a multi-set $S = \{s_1, \ldots, s_n\}$ we use the polynomial $(x-s_1)(x-s_2)\cdots(x-s_n)$, which we denote by $f_S(x)$. An important property of this representation is that a value y is in the set S if and only if $f_S(y) = 0$.

To hide the value of a polynomially-represented set, it is often useful to encrypt the set's polynomial using homomorphic encryption. Suppose we are given a polynomial $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, then the encryption of

[1] Based on our analysis of the THRESHOLD-PERFECT-HBC protocol given in [18]. An explicit protocol for the malicious model was not given, but by adding zero-knowledge proofs to the steps such a protocol could be constructed.

f, denoted by $E_{pk}(f)$, is the encryption, using E_{pk} on the coefficients of f, i.e., $E_{pk}(a_n), \ldots, E_{pk}(a_0)$. As described in previous work [9,18] when given $E_{pk}(f)$ it is possible to perform many operations on f. In this paper, we use the following operations:

1. *Polynomial Evaluation:* Given $E_{pk}(f)$, the public parameters of E_{pk} and a value x it is possible to compute $E_{pk}(f(x))$.
2. *Polynomial Addition:* Given $E_{pk}(f)$ and $E_{pk}(g)$ for two polynomials f and g, then it is possible to compute $E_{pk}(f + g)$.
3. *Polynomial Multiplication:* Given $E_{pk}(f)$ and g for two polynomials f and g it is possible to compute $E_{pk}(f * g)$.
4. *Polynomial Derivation:* Given $E_{pk}(f)$ it is possible to compute the encrypted polynomial of the derivative of f, i.e. it is possible to compute $E_{pk}(f')$.

4.3 Zero Knowledge Proofs

To extend our protocols to the malicious adversary model, we utilize zero knowledge proofs. In what follows, we outline the proofs of knowledge that are used in this paper. These proofs are similar to those used in [18] and can be efficiently realized using [2] and [5]. These proofs can be made non-interactive using the Fiat-Shamir heuristic [7].

In what follows, E_{pk} is a threshold additive-homomorphic encryption scheme.

1. $POPK(E_{pk}(x))$ represents a proof that the prover knows the plaintext x (i.e., it is a proof of plaintext knowledge). Furthermore, this proof can be done with $O(1)$ communication complexity.
2. *Proof of Correct Multiplication:* Given $E_{pk}(x)$ (where x may be unknown to the prover) and a value y, it is possible to publish values $E_{pk}(y)$ and $E_{pk}(z)$ and prove that $z = xy$. We denote this proof by $ZKPK(y|a = E_{pk}(y) \wedge b = E_{pk}(z) \wedge z = x * y)$. Furthermore, this proof can be done with $O(1)$ communication complexity.
3. *Proof of Correct Polynomial Evaluation:* When the prover is given $E_{pk}(f)$ for some polynomial f, then the prover can generate values $E_{pk}(x)$ for a known value x and $E_{pk}(z)$ along with a proof that $z = f(x)$. If the polynomials have degree n then, this proof requires $O(n)$ proofs of correct multiplication. We denote this proof by $ZKPK(x|y = E_{pk}(f(x)) \wedge z = E_{pk}(x))$.
4. *Proof of Correct Polynomial Multiplication:* When a prover is given $E_{pk}(f)$ for some polynomial f and another polynomial g, then the prover can generate $E_{pk}(g)$ and $E_{pk}(h)$ along with a proof that $h = f * g$. If the polynomials f and g have respective degree m and n, then this protocol requires $O(mn)$ proofs of correct multiplication. We denote this proof by $ZKPK(f|h = f * g \wedge y = E_{pk}(f))$.
5. *Proof of Correct Polynomial Construction:* If a prover has posted encryptions of a list of values $E_{pk}(x_1), \ldots, E_{pk}(x_n)$. Then the prover can post the encrypted polynomial $(x - x_1) \cdots (x - x_n)$ along with a proof that it was constructed properly. This requires $O(n^2)$ proofs of correct multiplication.

4.4 Mixes and Shuffles

In our protocols we use a cryptographic protocol for shuffling (e.g., mixing) a list of encrypted values. Specifically, a shuffle protocol uses list of encrypted values $E_{pk}(x_1), \ldots, E_{pk}(x_n)$ as input and the output of the protocol is another list of encrypted values $E_{pk}(y_1), \ldots, E_{pk}(y_n)$. Furthermore, the y-list is a permutation of the original x-list and any group of participants that is not a quorum (of the threshold encryption scheme) cannot associate a specific y-value back to a specific x-value. In our protocols we utilize a robust shuffling protocol where the participants obtain proof(in zero-knowledge) that the shuffle was performed correctly. Such a robust mix can be made from standard protocols [20,19,15]. We actually use a slight variation of a standard mix, in that our protocols shuffle tuples. That is, the input to the protocol is a list of tuples of encrypted values and the output is a permuted list of re-encrypted tuples that have a different tuple order, but the individual values inside of a tuple are in the same order. This can be achieved with techniques from [13]. The communication required to robustly mix n tuples with k participants is $O(k^2 n)$.

4.5 Bulletin Board

Our protocols use a bulletin board abstraction (i.e., all parties post information to a common area) that can be constructed using standard cryptographic techniques. We measure the communication requirements of our protocols as the amount of information posted on the bulletin board. It is worth noting that we do not need a robust bulletin board for our application (we are not trying to be secure against adversaries that try to force early termination). Thus our protocols could just utilize standard broadcast techniques.

5 Preliminary Protocols

In this section we give preliminary protocols for the honest-but-curious adversary model and the malicious adversary model. As a starting point we introduce a protocol for the two-party honest but curious adversary model, then we extend this to multiple parties, and we then extend it to the malicious model. It is worth noting that the malicious protocol is similar to the honest-but-curious protocol, but the protocols are presented as two separate protocols to enhance readability. As not to clutter this initial presentation, we postpone the discussion of several extensions to these protocols until the next section.

5.1 Two-Party Honest But Curious

In this section we propose a two-party protocol for set union in the honest but curious adversary model where only one party learns the result. This protocol is the most efficient such protocol to date that the authors are aware of, and requires only $O(n)$ communication. The main idea of this technique is as follows: Suppose that a participant has the encrypted polynomial, denoted by $E_{pk}(f_S)$

for some set S, then this participant can blindly evaluate this polynomial on each of his set items; we will denote a specific such value by $E_{pk}(f_S(s))$. Now $f_S(s) = 0$ if and only if $s \in S$ (with high probability). So if we create a tuple of the form: $(E_{pk}(f_S(s) * s)\ ;\ E_{pk}(f_S(s)))$, then this tuple will be $(0\ ;\ 0)$ if $s \in S$ and otherwise s can be recovered from the decrypted tuple values.

Thus a high level protocol for the two-party case is as follows: Participant P_1 encrypts his set as a polynomial and sends it to P_2 (using a homomorphic scheme that is chosen by P_1). P_2 then computes the above tuples and sends them back to P_1 in a random order. P_1 then decrypts the tuples to learn the values in S_2 that are not in S_1, these values are then added to the items in S_1 to produce the output of the protocol. The full description of the protocol is given in Figure 1.

Setup: Participant P_1 chooses a homomorphic encryption scheme, and denote the public encryption function by E_{pk}.

1. P_1 sends $E_{pk}(f_{S_1})$ along with the public parameters of E_{pk} to P_2.
2. For each value $s \in S_2$, P_2 chooses a random value r (chosen uniformly) and computes a tuple $(E_{pk}(f_{S_1}(s) * s * r)\ ;\ E_{pk}(f_{S_1}(s) * r))$. P_2 randomly permutes all of the tuples and sends them to P_1.
3. P_1 initially sets the output set to be S_1. For each tuple $(E_{pk}(x)\ ;\ E_{pk}(y))$ from the previous step, P_1 decrypts x and y. If both values are 0, then P_1 continues to the next tuple. Otherwise, P_1 adds $x * y^{-1}$ to the output set.

Fig. 1. Two-party HBC protocol for Set Union

Complexity Analysis: Step 1 of the protocol requires $O(n)$ communication and computation. It requires $O(n)$ computation to compute the value $f_{S_1}(s)$ for a single value s, and so Step 2 requires $O(n^2)$ computation. However, Step 2 requires only $O(n)$ communication. Finally, Step 3 requires $O(n)$ computation. Thus this protocol requires $O(n^2)$ computation, $O(n)$ communication, and $O(1)$ rounds. The computation can be reduced to $O(n \log\log n)$ using the bucketing techniques of [9][2].

Security Analysis: We must show that the communication transcript from this protocol is simulatable from the results of the protocol alone along with one of the participant's inputs. This is trivial to do in the case of P_2 since all communication from P_1 is encrypted with a semantically-secure homomorphic encryption scheme (where the private key is known only to P_1). The proof to show that the protocol is secure against P_1 is also straightforward, but is not as trivial. Suppose that a simulation algorithm has: $|S_2|$, S_1, and $S_1 \cup S_2$. Clearly, the simulator can compute $S_2 - S_1$ from the above information. The simulator then proceeds as follows: i) it computes $|S_2| - |S_2 - S_1|$ tuples of the form $(E_{pk}(0)\ ;\ E_{pk}(0))$, ii) it computes a tuple $(E_{pk}(s * r)\ ;\ E_{pk}(r))$ for a randomly chosen value r for each item $s \in S_2 - S_1$, and iii) it randomly permutes the tuples from the previous

[2] This requires minor modifications to the protocol.

two steps and outputs this as the simulated transcript. It is straightforward to show that this simulated transcript is indistinguishable from the values sent to P_1 in Step 2 of the protocol.

5.2 Multi-party Honest But Curious

In this section we introduce the preliminary protocol for the honest-but-curious adversary model for multiple participants. To compute the union of the individual sets, the protocol computes the multi-set union (using techniques of [18]) for each of the following sets of participants $\{P_1\}, \{P_1, P_2\}, \ldots, \{P_1, \ldots, P_n\}$. Then each participant P_i reports every item in S_i that is not in the multi-set union of $S_1, \ldots, S_{i-1}$ using the reporting technique from the previous section. The tuples are then mixed, to hide the source of each tuple, and then are decrypted to reveal the items in the set union. The full description of the protocol is given in Figure 2.

Complexity Analysis: In what follows we list the communication requirements of each step of the protocol:

1. *Step 1.a:* This requires $O(n)$ communication for each participant, and thus this requires $O(kn)$ total communication.
2. *Step 1.b:* This requires $O(in)$ communication for participant P_i, and thus this requires $O(k^2n)$ total communication.
3. *Step 1.c*: Each participant has to post $O(n)$ tuples, and thus this requires $O(kn)$ total communication.
4. *Step 2.b:* Each participant has to post $O(kn)$ tuples, and thus this requires $O(k^2n)$ total communication.
5. *Step 3:* Since we are mixing $O(kn)$ values in a non-robust manner this requires $O(k^2n)$ total communication.
6. *Step 4:* Each decryption requires $O(k)$ communication, and so this requires $O(k^2n)$ total communication.

In summary, this protocol requires $O(k^2n)$ communication and $O(k)$ rounds.

Security Analysis: We must show that the communication transcript from this protocol is simulatable from the results of the protocol alone. This is a relatively straight-forward simulation, and so we are a bit informal throughout this discussion. Suppose that a simulation algorithm is given $\eta_1, \ldots, \eta_k$ along with $\cup_{i=1}^{k} S_i$. Now the simulation algorithm can easily create k simulation sets $SS_1, \ldots, SS_k$ such that $|SS_i| = \eta_i$ and $\cup_{i=1}^{k} SS_i = \cup_{i=1}^{k} S_i$. Now the simulation algorithm simply mimics the protocol in Figure 2 (for the shuffling phase it acts as all of the mix servers) and outputs the transcript of this session. We claim that this simulated transcript is indistinguishable from transcript generated by the real protocol to any adversary that does not establish a quorum of participants. First, in steps 1 and 2 everything is encrypted with a semantically-secure cryptosystem so these values will be indistinguishable. Step 3 is indistinguishable because both are runs of a mix protocol. Finally, Step 4 is indistinguishable because both are randomly permuted lists tuples, and we claim that decrypted tuples that reveal a set value

Setup: The participants have agreed on a threshold Homomorphic encryption scheme, and denote the public encryption function by E_{pk}.

1. *Build tuples:*
   (a) *Post polynomial representation of sets:* Participant P_i posts $E_{pk}(f_{S_i})$ to the bulletin board.
   (b) *Post polynomials:* Participant P_i (for $i = 2, \ldots, k$) posts $E_{pk}(\prod_{j=1}^{i}(f_{S_j}))$ to the bulletin board.
   (c) *Post tuples:* Participant P_i posts the following tuples to the bulletin board:
      i. P_1 posts $(E_{pk}(s)\ ;\ (E_{pk}(1))$ for each value $s \in S_1$.
      ii. Participants P_i (for $i = 2, \ldots, k$) posts tuples for every $s \in S_i$ as follows: $\left(E_{pk}(\prod_{j=1}^{i-1}(f_{S_j}(s) * s)\ ;\ E_{pk}(\prod_{j=1}^{i-1}(f_{S_j}(s))\right)$.
2. *Randomize tuple values:* For each tuple in $(E_{pk}(x)\ ;\ E_{pk}(y))$ that was posted by any participant in the previous step, the participants do the following (Note that this step can be done in parallel for all tuples):
   (a) Each participant, P_i chooses a non-zero random values r_i chosen uniformly.
   (b) The participants multiply the tuple's values by their random value. That is, P_i posts $\left(E_{pk}(x * \prod_{j=1}^{i} r_j)\ ;\ E_{pk}(y * \prod_{j=1}^{i} r_j)\right)$ to the bulletin board. Note that P_i must wait until P_{i-1} has posted his tuples before P_i can post his tuples.
3. *Shuffle:* The parties engage in a secure shuffle protocol for all of the tuples generated by P_k in the previous step.
4. *Decrypt results:* For each tuple $(E_{pk}(x)\ ;\ E_{pk}(y))$, the parties jointly decrypt x and y. If both values are 0, then the parties continue to the next tuple. Otherwise, the parties add $x * y^{-1}$ to the output set.

Fig. 2. Multi-party HBC protocol for Set Union

s are indistinguishable from a tuples $(s * r\ ;\ r)$ for some random value r. This follows from Step 2 of the protocol, because as long as one participant is honest the values will be multiplied by a random value unknown to the adversary.

5.3 Malicious Model

In this section we introduce the protocol for the malicious model. This protocol is similar to the honest-but-curious protocol, however there are a few crucial differences. The main difference is that the parties commit to their set values and then at each step of the protocol the parties prove in zero knowledge that they are following the protocol correctly. The full description of the protocol is in Figure 3.

Complexity Analysis: In what follows we list the communication requirements of each step of the protocol:

1. *Step 1:* Each participant has to post $O(n)$ values, and thus the total communication is $O(kn)$.

Setup: The participants have agreed on a threshold Homomorphic encryption scheme, and denote the public encryption function by E_{pk}.

1. *Commit to sets:* Participant P_i posts the following values to the bulletin board $E_{pk}((S_i)_1), \ldots, E_{pk}((S_i)_{\eta_i})$ along with a proof of plaintext knowledge.
2. *Build tuples:*
   (a) *Post polynomial representation of sets:* Participant P_i posts $E_{pk}(f_{S_i})$ to the bulletin board along with a proof of correct polynomial construction (using the commitments from the previous step).
   (b) *Post polynomials:* Participant P_i (for $i = 2, \ldots, k$) posts $E_{pk}(\prod_{j=1}^{i}(f_{S_j}))$ to the bulletin board along with a proof of correct polynomial multiplication.
   (c) *Post tuples:* Participant P_i posts the following tuples to the bulletin board:
      i. P_1 posts $(E_{pk}(s)\ ;\ (E_{pk}(1))$ for each value $s \in S_1$ along with a proof of correct construction.
      ii. Participants P_i (for $i = 2, \ldots, k$) posts tuples for every $s \in S_i$ as follows: $\left(E_{pk}(\prod_{j=1}^{i-1}(f_{S_j}(s) * s)\ ;\ E_{pk}(\prod_{j=1}^{i-1}(f_{S_j}(s))\right)$ along with zero knowledge proofs that this tuple is formed correctly. Specifically, the participant posts a proof of correct polynomial evaluation and a proof of correct multiplication.
3. *Randomize tuple values:* For each tuple in $(E_{pk}(x)\ ;\ E_{pk}(y))$ that was posted by any participant in the previous step, the participants do the following (Note that this step can be done in parallel for all tuples):
   (a) Each participant, P_i chooses a random values r_i. P_i also posts a commitment of this random value $E_{pk}(r_i)$ to the bulletin board along with a proof that r_i is non-zero.
   (b) The participants multiply the tuple's values by their random value. That is, P_i posts $\left(E_{pk}(x * \prod_{j=1}^{i} r_j)\ ;\ E_{pk}(y * \prod_{j=1}^{i} r_j)\right)$ to the bulletin board along with a proof of correct construction. Note that P_i must wait until P_{i-1} has posted his tuples before P_i can post his tuples.
4. *Shuffle:* The parties engage in a secure shuffle protocol for all of the tuples generated in the previous step by P_k.
5. *Decrypt results:* For each tuple $(E_{pk}(x)\ ;\ E_{pk}(y))$, the parties jointly decrypt x and y. If both values are 0, then the parties continue to the next tuple. Otherwise, the parties add $x * y^{-1}$ to the output set.

Fig. 3. Malicious protocol for Set Union

2. *Step 2.a:* Each participant has to post $O(n^2)$ data (the size of the correct polynomial construction proof), and thus the total communication is $O(kn^2)$.
3. *Step 2.b*: This requires $O(in^2)$ communication for participant P_i (who must do a proof of correct polynomial multiplication between a polynomial of size $O(in)$ and a polynomial of size $O(n)$), and thus this requires $O(k^2n^2)$ total communication.
4. *Step 2.c:* This requires $O(in^2)$ communication for participant P_i (who must do a proof of correct polynomial evaluation for a polynomial of size $O(in)$ on $O(n)$ values), and thus this requires $O(k^2n^2)$ total communication.

5. *Step 3.a:* This requires each participant to post $O(kn)$ communication, and thus the total communication is $O(k^2n)$.
6. *Step 3.b:* This requires each participant to post $O(kn)$ communication, and thus the total communication is $O(k^2n)$.
7. *Step 4:* This requires a robust mix of $O(kn)$ values with k participants. Thus this requires $O(k^3n)$ total communication.
8. *Step 5:* Each decryption requires $O(k)$ communication, and so this requires $O(k^2n)$ total communication.

In summary, this protocol requires $O(k^2n^2 + k^3n)$ communication and $O(k)$ rounds.

Security Analysis: In this section we give an argument for security for the malicious model protocol. All of the communication in steps 1-4 are either values encrypted with a homomorphic encryption scheme or are zero knowledge proofs. It is easy to verify that the zero knowledge proofs do not reveal anything other than predicates of the form: was this step done properly. Thus all of these steps can easily be simulated in a way that is indistinguishable from the real transcript. However, we must show that the decrypted values in Step 5 of the protocol can be simulated from the output of the protocol in a manner that is indistinguishable from the values in the protocol (even if a group of participants deviates from the protocol). If any of the zero knowledge proofs fail, then the protocol terminates and Step 5 is not reached, and so in what follows we assume that the zero knowledge proofs have all passed.

In Step 1 of the protocol the participants submit of list of committed values. Clearly, this same set of values could be injected into the ideal model (assuming that the ideal model allows multi-set inputs). We must show that the simulator with the result from the ideal model will produce a list of values that are indistinguishable from these values. To do this, we first define the simulation algorithm. Suppose that $S = \bigcup_{i=1}^{k} S_i$, and that $N = \sum_{i=1}^{k} |S_i|$. The simulation then proceeds as follows: i) it produces $N - |S|$ tuples of the form $(0\ ;\ 0)$ and for each value $s \in S$ it creates a tuple of the form $(s * r\ ;\ r)$ for a randomly chosen value r. The simulation algorithm randomly permutes these N tuples and outputs this as the transcript for Step 5.

The following list enumerates the state of the N tuples that are produced in each of the steps of the protocol

1. *Step 2:* Participant P_i's jth tuple will be $(0\ ;\ 0)$ if $(S_i)_j$ is in one of the sets $S_1, \ldots, S_{i-1}$. Otherwise, it will be $\left((S_i)_j * v\ ;\ v\right)$ for a value v that may reveal information.
2. *Step 3:* This step multiplies the values in each tuple by a random non-zero value using a standard protocol (i.e., everyone multiplies it by their own random non-zero value). Thus participant P_i's jth tuple will be $(0\ ;\ 0)$ if $(S_i)_j$ is in one of the sets $S_1, \ldots, S_{i-1}$. Otherwise, it will be $\left((S_i)_j * r\ ;\ r\right)$ for a randomly chosen value r. Thus at the end of this protocol there will be $N - |S|$ tuples of the form $(0\ ;\ 0)$ and for each value $s \in S$ it creates a

tuple of the form $(s * r\ ;\ r)$ for a randomly chosen value r. This is just like the simulation algorithm except that the order of the tuples in the protocol at the end of this step reveals information.

3. *Step 4:* The tuples are randomly shuffled with a robust mix. Thus, the tuples will be the same as in the previous step but are in a random order.

6 Extensions

In this section we introduce various extensions of the preliminary protocols in the previous section, including: padding sets, computing the cardinality, countering the empty-set attack, and computing the over-threshold set union.

6.1 Padding

As mentioned earlier, the preliminary protocols leak the number of values that each party has in their set. In this section we introduce a method for padding a list to obfuscate this value, and thus all that is revealed is an upper bound on the cardinality of each party's set. Suppose that a participant P_i wants to report a set S_i and a size η_i where $\eta_i \geq |S_i|$. In the HBC protocol P_i would make the following changes: i) The participant would use a polynomial $g_{S_i,\eta_i} = f_{S_j} * x^{\eta_i - |S_i|}$ and would use this as his polynomial in Step 2 and ii) when reporting his "dummy" values in Step 3 of the protocol the participant posts $(E_{pk}(0)\ ;\ E_{pk}(0))$.

The changes to the malicious protocol are a little more involved (to prevent the participants from failing a zero knowledge proof). The main change is that the subject will commit to a larger set of values where the dummy values set to a value not in $\mathcal{U}$ and a dummy item's random hiding factor in Step 3 is set to 0. It is also required that the proofs that a random values are non-zero are removed for a participant's own values. The rest of the protocol then remains unchanged.

6.2 Cardinality

There are some situations where the goal is to reveal only the cardinality of the set union. It is relatively straight-forward to modify the preliminary protocols to compute the cardinality. In Step 1 (of the HBC protocol) and Step 2 (in the malicious protocol) all that needs to be changed is that when creating the tuples, participant P_i used 1 instead of $(S_i)_j$. Now, when the tuples are decrypted duplicate items will still be $(0\ ;\ 0)$ but first-time items will be $(r\ ;\ r)$ (for some random value r). Thus, the number of tuples that are not $(0\ ;\ 0)$ is the cardinality of the set union.

6.3 Empty-Set Attack

It is possible for a malicious party to set their set to the empty-set. This is particularly damaging when there are only two participants, as this will

reveal the honest participant's exact set. We now outline several strategies for countering this attack:

1. A simple solution is to require that each participant's set is not the empty-set. One way of doing this is to make sure that the leading coefficient of the polynomial is non-zero. However, this has limited effectiveness, because the adversary can use n items that are very infrequent as their input set. Or worse, the adversary could use a non-decomposable polynomial (i.e., one that is never 0, for the protocol).
2. A more complicated approach would be to make sure that each party's set has certain properties. However, these requirements would vary from domain to domain, and so each new domain would require a separate protocol for determining if a set is valid. Thus, this is not a general solution.
3. Another approach is for participants to require some overlap between the other participant's set and their own. Thus in the two party-case the participants would first engage in a cardinality of set intersection protocol and would continue only if this cardinality was over a threshold. This check can be done without revealing the actual cardinality. This is not a perfect approach, because an adversary can still test if an honest participant has a specific item (or small set of items), but it does help prevent complete revelation of the honest party's set in a single run of the protocol.

6.4 Over-Threshold Set Union

In this section we introduce a protocol for computing all items that appear t or more times. This protocol does not reveal how many times an item appears (even if it is in the result). A similar protocol was given in [18], but the given protocol reveals how many times an item in the result appeared. Of course, many of the ideas from [18] could be combined to make such a protocol, we believe that the following protocol will be more efficient than such a protocol. However, some of the ideas in the following protocol were also presented in [18] (specifically taking the $(t-1)$th derivative to determine if an item has appeared t or more times). We present this protocol for the honest-but-curious model in Figure 4; a malicious model protocol will be given in the full version of the paper[3].

The main idea behind this protocol is that each participant first learns which of its items appears t or more times, and then these values are used in a standard set union protocol to merge the items and to hide the multiplicity of the items. Clearly, this intermediate result is simulateable from the output alone (i.e., given all items that appear t or more times a participant can compute which items in its set appear t or more times). To compute which items are in t or more sets the participants compute the $(t-1)$th derivative of the polynomial for the multi-set union of every participant's set. Note that when this polynomial evaluates to 0 for a specific set item then the item will have appeared t or more times with high probability.

[3] It is worth noting that the protocol in Figure 4 would not work by simply adding zero knowledge proofs, because this would not prevent a party from submitting multiple copies of the same value.

Setup: The participants agree on a threshold Homomorphic encryption scheme, and denote the public encryption function by E_{pk}.

1. *Compute multi-set union*: Using Step 1 of the protocol in Figure 2 the participants compute $E_{pk}(\prod_{j=1}^{k}(f_{S_j}))$, which we denote my $E_{pk}(m)$.
2. The participants then compute the value $E_{pk}(m^{(t-1)})$ (i.e., the $(t-1)$th deriviative of the multi-set union).
3. For each value $s \in S_i$ participant P_i posts the following tuple $\left(E_{pk}(m^{(t-1)}(s)) \; ; \; E_{pk}(s)\right)$ to the bulletin board.
4. Using Step 2 of the protocol in Figure 2 the participants multiply the first value of each tuple by a random value and post the new values to the bulletin board.
5. For each tuple $\left(E_{pk}(m^{(t-1)}(s) * r) \; ; \; E_{pk}(s)\right)$ that was posted in the previous step (the participants compute $E_{pk}((m^{(t-1)}(s) * r) + s)$ and jointly decrypt this value so that only the participant that contributed the value will learn the plaintext.
6. For each item that a participant posted in Step 3 they obtain either the value itself or a random value. Each participant then builds a new set of the items that survived elimination and pads the list to its original size. The participants then engage in a privacy-preserving protocol for Set Union with these new sets.

Fig. 4. HBC protocol for Over-Threshold Set Union

7 Conclusions

In this paper we introduced protocols for privacy-preserving set union that are more efficient than previous such protocols. We believe that these new protocols will have many applications in data mining and other domains. We also introduced a new over-threshold set union protocol.

Acknowledgements

The author would like to thank the anonymous reviewers for their comments and useful suggestions.

References

1. J. Brickel and V. Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *Proceedings of Advances in Cryptology - ASIACRYPT '05*, volume 3788 of *Lecture Notes in Computer Science*, pages 236–252, 2005.
2. R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Proceedings of Advances in Cryptology - EUROCRYPT '01*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–299, 2001.
3. I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Proceedings of Advances in Cryptology - CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–411, 2005.

4. I. Damgard and M. Jurik. Efficient protocols based on probabilistic encryption using composite degree residue classes, 2000.
5. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 119–136, London, UK, 2001. Springer-Verlag.
6. A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 211–222, New York, NY, USA, 2003. ACM Press.
7. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings of Advances in Cryptology - CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, 1986.
8. Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. *Lecture Notes in Computer Science*, 1962: 90–104, 2001.
9. M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proceedings of Advances in Cryptology - EUROCRYPT '04*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19, 2004.
10. O. Goldreich. *Foundations of Cryptography: Volume II Basic Application.* Cambridge University Press, 2004.
11. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM Press.
12. S. Goldwasser and S. Micali. Probabilistic encryption, 1984.
13. P. Golle and M. Jakobsson. Reusable anonymous return channels. In *WPES '03: Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, pages 94–100, New York, NY, USA, 2003. ACM Press.
14. B. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 78–86, New York, NY, USA, 1999. ACM Press.
15. M. Jakobsson, A. Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of USENIX'02*, pages 339–353, 2002.
16. M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1026–1037, September 2004.
17. A. Kiayias and A. Mitrofanova. Testing disjointness of private datasets. In *Proceedings of Financial Cryptography*, volume 3570 of *Lecture Notes in Computer Science*, pages 109–124, 2005.
18. L. Kissner and D. Song. Privacy-preserving set operations. In *Proceedings of Advances in Cryptology - CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, 2005. Full version appears at `http://www.cs.cmu.edu/ leak/`.
19. C.A. Neff. A verifiable secret shuffle and its application to e-voting. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125, New York, NY, USA, 2001. ACM Press.
20. W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *Proceedings of ICICS '97*, volume 1334 of *Lecture Notes in Computer Science*, pages 440–444, 1997.

21. C. O'Keefe, M. Yung, L. Gu, and R. Baxter. Privacy-preserving data linkage protocols. In *WPES '04: Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 94–102, New York, NY, USA, 2004. ACM Press.
22. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 573–584, 1999.
23. J. Vaidya and C. Clifton. Privacy - preserving top-k queries. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, pages 545–546, 2005.
24. A. Yao. How to generate and exchange secrets. In *Proceedings of FOCS*, pages 162–167, 1986.