Secure Computation of the k^{th}-Ranked Element

Gagan Aggarwal[1]⋆, Nina Mishra[2]⋆⋆, and Benny Pinkas

[1] Computer Science Department, Stanford University
gagan@cs.stanford.edu
[2] HP Labs and Stanford University
nmishra@hpl.hp.com, nmishra@theory.stanford.edu
[3] HP Labs, Princeton
benny.pinkas@hp.com, benny@pinkas.net

Abstract. Given two or more parties possessing large, confidential datasets, we consider the problem of securely computing the k^{th}-ranked element of the union of the datasets, e.g. the median of the values in the datasets. We investigate protocols with sublinear computation and communication costs. In the two-party case, we show that the k^{th}-ranked element can be computed in $\log k$ rounds, where the computation and communication costs of each round are $O(\log M)$, where $\log M$ is the number of bits needed to describe each element of the input data. The protocol can be made secure against a malicious adversary, and can hide the sizes of the original datasets. In the multi-party setting, we show that the k^{th}-ranked element can be computed in $\log M$ rounds, with $O(s \log M)$ overhead per round, where s is the number of parties. The multi-party protocol can be used in the two-party case and can also be made secure against a malicious adversary.

1 Introduction

For a set $S \subset \mathbb{R}$, the k^{th}*-ranked element* is the value $x \in S$ that is ranked k in the list S sorted in increasing order of all elements. The p^{th}*-percentile* is the value $x \in S$ such that $p\%$ of the values in S are below x. Of particular interest is the median or 50th-percentile, which is the element with rank $p = \lceil |S|/2 \rceil$. Given two parties A and B with datasets $D_A, D_B \subset \mathbb{R}$, respectively, we consider the problem of privately computing the k^{th}-ranked element of $D_A \cup D_B$. We also consider this problem in the multi-party case. In the setting we consider, the datasets D_A and D_B contain proprietary information, thus neither party is willing to share its data with the other. In addition, we assume that $n = |D_A| + |D_B|$ is very large.

There are many situations where secure computation of the k^{th}-ranked element is useful. For example, two health insurance companies may wish to compute the median life expectancy of their insured smokers. In such a setting, both the number of insured smokers as well as their life expectancies are private information, but the median life

⋆ Supported in part by a Stanford Graduate Fellowship, NSF Grant ITR-0331640 and NSF Grant EIA-0137761.

⋆⋆ Supported in part by NSF grant EIA-0137761.

expectancy is of combined mutual interest. Another example is the annual Taulbee survey which collects salary and demographic data for faculty in computer science and computer engineering departments in North America. Each year, academic departments report only a small number of statistics like the average salary for assistant, associate and full professor positions. The Taulbee survey is thus able to publish only limited aggregate information. A secure, multi-party solution for the k^{th}-ranked element would enable universities to quickly compute the median salary without trusting individual salaries to Taulbee. Finally, secure computation of the k^{th}-ranked element facilitates secure computation of histograms [9,16,12].

The problem we discuss is exactly that of secure computation. Namely, it involves several parties with private inputs that wish to compute a function of their joint inputs, and require that the process of computing the function does not reveal to an adversarial party (or a coalition of such parties) any information that cannot be computed using the input of the adversary and the output of the function.

There exist well known solutions for secure computation of any function (see e.g. [18,11]). The general method employed by these solutions is to construct a combinatorial circuit that computes the required function, and then run a distributed protocol that securely evaluates the circuit.[4] The communication overhead of these generic protocols is linear in the size of the circuit. The computation involves (at the least) running an oblivious transfer protocol for every input gate, or for every gate of the circuit, depending on the implementation. The k^{th}-ranked element can be computed via a circuit of size $\Omega(n \log M)$ (since reading in the input requires at least $n \log M$ gates), which implies that for large values of n the overhead of a secure protocol that is constructed by generic constructions is too large. In another generic construction, Naor and Nissim [15] show that any two-party communication protocol can be translated into a secure computation protocol, such that a protocol with communication complexity of c bits is transformed to a secure protocol with overhead of 2^c public key operations. This transformation can be applied to a protocol, due to Karchmer, for computing the median with $\log n$ communication bits [13].

Contributions We are motivated by applications where the total number of points owned by the parties (n) is very large, and thus even a linear communication and computation overhead might be prohibitive. Thus, we describe protocols with sublinear communication and computation overhead. Specifically, in the two-party case, we reduce the computation of the k^{th}-ranked element to $O(\log k)$ secure comparisons of ($\log M$)-bit inputs[5], where $\log M$ is the number of bits needed to describe the elements in the sets D_A, D_B. We also show how to obtain security against malicious adversaries. In the multi-party case, we reduce the computation of the k^{th}-ranked element to $O(\log M)$ simple secure computations that involve additions and a comparison of ($\log M$)-bit long numbers. Again, this protocol can be made secure against malicious adversaries. Inter-

[4] The interested reader can find a description of these protocols in the references above. Alternatively, descriptions of the two-party protocols are available at, e.g., [14,10], and descriptions of the multi-party protocols can be found, for example, in [1,8,10].

[5] If the two parties possess inputs x and y, a *secure comparison* reveals 0 if $x \geq y$ and 1 otherwise, and nothing more, assuming the usual cryptographic assumptions.

estingly, the multi-party solution can be applied to the two-party scenario if it uses secure two-party protocols as primitives. The protocol can even be directly applied to inputs that contain duplicate items, whereas the two-party protocol requires inputs comprising of distinct inputs. This is in contrast to the typical case in secure computation where secure multi-party protocols require the presence of an honest majority, which is not available in the two-party case.

The protocols given in this paper are modifications of well known algorithms in the communication complexity literature [17,13]. Our contribution is the modifications and proofs of security that result in privacy-preserving solutions, for both semi-honest and malicious adversaries. In addition, we show how the parties can compute the k^{th}-ranked element while hiding from each other the actual sizes of their databases.

Efficient Secure Computation via Reduction and Composition We take the same approach as that of previous solutions for secure computation of large inputs (e.g. [14,6,4]), and reduce this task to many invocations of secure computation of simpler functions of small inputs (but unlike these constructions, we also design protocols which are secure against malicious adversaries). That is, we describe a protocol for computing the k^{th}-ranked value which uses oracle queries to a few simple functionalities and is secure if these functionalities are computed by a trusted oracle. A composition theorem (see [2,3] and discussions below) shows that if the oracle queries are replaced by secure protocols, then the resulting combined protocol is also secure. In the semi-honest case the oracle queries can be replaced by very simple invocations of secure function evaluation. In the malicious adversary case they are replaced by a reactive secure computation of a simple function. We also note that the protocol computes the *exact* value of the k^{th}-ranked item, rather than computing an approximation as in [6].

1.1 Security Definitions and a Composition Theorem

We describe protocols that are secure against malicious adversaries. We therefore use definitions that compare the actual execution of the protocol to an "ideal" implementation, rather than use definitions that use simulation. The definitions we use follow those of Canetti and of Goldreich [2,10]. We also state a composition theorem that is used in analyzing the security of the protocols.

A *semi-honest adversary* is an adversary that follows the instructions defined by the protocol. It might try, however, to use the information that it learns during the execution in order to learn information about the inputs of the other parties. A *malicious adversary* is an adversary that can behave arbitrarily. In particular, there are several things that a malicious adversary can do which we cannot hope to avoid: (1) it can refuse to participate in the protocol, (2) it can substitute an arbitrary value for its input, and (3) it can abort the protocol prematurely. Following [2,10] we do not consider here solutions for the fairness of the protocol (i.e. item (3) above – the early termination problem) since there is no perfect solution for this issue and existing solutions are quite complex.

The security definition we use captures both the correctness and the privacy of the protocol. We only provide definitions for the two-party case. The definition is based on a comparison to the ideal model with a trusted third party (TTP), where corrupt parties can choose to give an arbitrary input to the trusted party, and to terminate the protocol prematurely, even at a stage where they have received their output and the other parties have not. We limit it to the case where both parties compute the same function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$.

Definition 1 (The Ideal Model). *A strategy for party A in the ideal model is a pair of PPT (probabilistic polynomial time) algorithms, $A_I(X,r)$ that uses the input X and a sequence of coin flips r to generate an input that A sends to the trusted party, and $A_O(X,r,Z)$ which takes as an additional input the value Z that A receives from the TTP, and outputs A's final output. If A is honest then $A_I(X,r) = X$ and $A_O(X,r,Z) = Z$. A strategy for party B is similarly defined using functions $B_I(Y,r)$ and $B_O(Y,r,Z)$.*

The definition is limited to the case where at least one of the parties is honest. We call an adversary that corrupts only one of the parties an admissible adversary. *The joint execution of A and B in the ideal model, denoted $\mathsf{IDEAL}_{A,B}(X,Y)$, is defined to be*

- *If B is honest,*
  - $\mathsf{IDEAL}_{A,B}(X,Y)$ *equals* $(A_O(X,r,f(X',Y)), f(X',Y))$, *where* $X' = A_I(X,r)$ *(in the case that A did not abort the protocol),*
  - *or,* $\mathsf{IDEAL}_{A,B}(X,Y)$ *equals* $(A_O(X,r,f(X',Y)), -)$, *where* $X' = A_I(X,r)$ *(if A terminated the protocol prematurely).*
- *If A is honest*
  - $\mathsf{IDEAL}_{A,B}(X,Y)$ *equals* $(f(X,Y'), B_O(Y,r,f(X,Y')))$, *where* $Y' = B_I(Y,r)$,
  - *or,* $\mathsf{IDEAL}_{A,B}(X,Y)$ *equals* $(-, B_O(Y,r,f(X,Y')))$, *where* $Y' = B_I(Y,r)$.

In the real execution a malicious party could follow any strategy that can be implemented by a PPT algorithm. The strategy is an algorithm mapping a partial execution history to the next message sent by the party in the protocol.

Definition 2 (The Real Model (for semi-honest and malicious adversaries)). *Let f be as in Definition 1, and Π be a two-party protocol for computing f. Let (A',B') be a pair of PPT algorithms representing the parties' strategies. This pair is admissible w.r.t. Π if at least one of (A',B') is the strategy specified by Π for the corresponding party. In the* semi-honest case *the other party could have an arbitrary output function. In the* malicious *case, the other party can behave arbitrarily throughout that protocol.*

The joint execution of Π in the real model, denoted $\mathsf{REAL}_{\Pi,A',B'}(X,Y)$ is defined as the output pair resulting from the interaction between $A'(X)$ and $B'(Y)$.

The definition of security states that an execution of a secure real model protocol under any admissible adversary can be simulated by an admissible adversary in the ideal model.

Definition 3 (Security (for both the semi-honest case and the malicious case)). *Let f and Π be as in Definition 2. Protocol Π* **securely computes** *f if for every PPT pair (A',B') that is admissible in the real model (of Definition 2) there is a PPT pair (A,B) that is admissible in the ideal model (of Definition 1), such that $\mathsf{REAL}_{\Pi,A',B'}(X,Y)$ is computationally indistinguishable from $\mathsf{IDEAL}_{A,B}(X,Y)$.*

Reactive Computations A reactive computation consists of steps in which parties provide inputs and receive outputs. Each step generates a *state* which is used by the following step. The input that a party provides at step i can depend on the outputs that it received in previous steps. (We limit ourselves to synchronous communication, and to an environment in which there are secure channels between the parties.) The protocols that we design for the malicious case implement reactive computation. Security definitions and constructions for reactive computation were discussed in [3,5] (in particular, they enable parties to abort the protocol at arbitrary stages). We will not describe these definitions in this extended abstract, due to their length and detail.

A Composition Theorem Our protocols implement the computation of the k^{th}-ranked element by running many invocations of secure computation of simpler functionalities. Such constructions are covered by theorems of secure composition [2,3]. Loosely speaking, consider a *hybrid model* where the protocol uses a trusted party that computes the functionalities $f_1, \ldots, f_\ell$. The secure composition theorem states that if we consider security in terms of comparing the real computation to the ideal model, then if a protocol is secure in the hybrid model, and we replace the calls to the trusted party by calls to secure protocols computing $f_1, \ldots, f_\ell$, then the resulting protocol is secure. A secure composition theorem applies to reactive computation, too [3,5].

2 Two-Party Computation of the k^{th} Element

This section describes protocols for secure two-party computation of the k^{th}-ranked element of the union of two databases. The protocols are based on the observation that a natural algorithm for computing the k^{th}-ranked element discloses very little information that cannot be computed from the value of the k^{th}-ranked element itself. Some modification to that protocol can limit the information that is leaked by the execution to information that can be computed from the output alone.

To simplify the description of the basic, insecure, protocol, we describe it for the case of two parties, A and B, each of which has an input of size $n/2$, that wish to compute the value of the median, i.e. $(n/2)^{th}$-ranked element, of the union of their two inputs sorted in increasing order of their values. This protocol is a modification of the algorithm given in [17,13]. Assume for simplicity that all input values are different. The protocol operates in rounds. In each round, each party computes the median value of his or her input, and then the two parties compare their two median values. If A's median value is smaller than B's then A adjusts her input by removing the values which are less than or equal to her median, and B removes his input items which are greater than his median. Otherwise, A removes her items which are greater than her median and B removes his items which are less than or equal to his median. The protocol continues until the inputs are of length 1 (thus the number of rounds is logarithmic in the number of input items). The protocol is correct since when A's median is smaller than B's median, each of the items that A removes is smaller than A's median, which is smaller than at least $n/4$ inputs of A and $n/4$ inputs of B. Therefore the removed item cannot be the median. Also, the protocol removes $n/4$ items which are smaller than the median and $n/4$ which are greater than it, and therefore the median of the new data is the same as that of the original input. Other cases follow similarly.

Suppose now that the comparison is done privately, i.e. that the parties only learn whose median value is greater, and do not learn any other information about each others median value. We show below that in this case the protocol is secure. Intuitively this is true since, e.g., if party A knows the median value of her input and the median of the union of the two inputs, and observes that her median is smaller than the median of the union, then she can deduce that her median value is smaller than that of B. This means that given the final output of the protocol party A can simulate the results of the comparisons. Consequently, we have a reduction from securely computing the median of the union to securely computing comparisons.

Secure comparison: The main cryptographic primitive that is used by the protocol is a two-party protocol for secure comparison. The protocol involves two parties, where party A has an input x and party B has an input y. The output is 0 if $x \geq y$ and 1 otherwise. The protocol (which essentially computes a solution to Yao's millionaires problem) can be implemented by encoding the comparison function as a binary circuit which compares the bits encoding the two inputs, and applying to it Yao's protocol for secure two-party computation. The overhead is $|x|$ oblivious transfers, and $O(|x|+|y|)$ applications of a pseudo-random function, as well as $O(|x|+|y|)$ communication. More efficient, non-interactive comparison protocols also exist (see e.g. [7]).

2.1 A Protocol for Semi-Honest and Malicious Parties

Following is a description of a protocol that finds the k^{th}-ranked element in the union of two databases and is secure against semi-honest parties. The computation of the median is a specific case where k is set to be the sum of the two inputs divided by two. The protocol reduces the general problem of computing the k^{th}-ranked element of arbitrary size inputs, to the problem of computing the median of two inputs of equal size, which is also a power of 2. To simplify the exposition, we assume that all the inputs are *distinct*. This issue is further discussed later.

Security against a malicious adversary. The protocol for the semi-honest case can be amended to be secure against malicious adversaries. The main change is that the protocol must now verify that the parties provide consistent inputs to the different invocations of the secure computation of the comparisons. For example, if party A gave an input of value 100 to a secure comparison computation, and the result was that A must delete all its input items which are smaller than 100, then A cannot provide an input which is smaller than 100 to any subsequent comparison. We provide a proof that given this enforcement, the protocol is secure against malicious behavior. For this protocol, we do not force the input elements to be integers. However, if such an enforcement is required (e.g. if the input consists of rounded salary data), then the protocol for the malicious case verifies that there is room for sufficiently many distinct integers between the reported values of different elements of the input. This is made more precise later.

In protocol FIND-RANKED-ELEMENT that we describe here, we specify the additional *functionality* that is required in order to ensure security against malicious parties. Then in Section 2.3 we describe how to implement this functionality, and prove that given this functionality the protocol is secure against malicious adversaries. Of course, to obtain a protocol which is only secure against semi-honest adversaries, one should ignore the additional highlighted steps that provide security in the malicious case.

Protocol FIND-RANKED-ELEMENT
Input: D_A known to A, and D_B known to B. Public parameter k (for now, we assume that the numerical value of the rank of the element is known). All items in $D_A \cup D_B$ are distinct.
Output: The k^{th}-ranked element in $D_A \cup D_B$.

1. Party A (resp., B) initializes S_A (resp., S_B) to be the sorted sequence of its k smallest elements in D_A (resp., D_B).
2. If $|S_A| < k$ then Party A pads $(k - |S_A|)$ values of "+∞" to its sequence S_A. Party B does the same: if $|S_B| < k$ then it pads $(k - |S_B|)$ values of "+∞" to its sequence S_B.
3. Let 2^j be the smallest power of 2 greater than or equal to k. Party A pre-pads S_A with $(2^j - k)$ values of "-∞" and Party B pads S_B with $(2^j - k)$ values of "+∞". (The result is two input sets of size 2^j each, whose median is the k^{th}-ranked element in $D_A \cup D_B$.)

   In the malicious case: The protocol sets bounds $l_A = l_B = -\infty$ and $u_A = u_B = \infty$.
4. For $i = (j-1), \ldots, 0$:
   A. A computes the $(2^i)^{th}$ element of S_A, denoted m_A, and B computes the $(2^i)^{th}$ element of S_B, m_B. (I.e., they compute the respective medians of their sets.)
   B. A and B engage in a *secure computation* which outputs 0 if $m_A \geq m_B$, and 1 if $m_A < m_B$.
      In the malicious case: The secure computation first checks that $l_A < m_A < u_B$ and $l_B < m_B < u_B$. If we want to force the input to be integral, then we check that $l_A + 2^i < m_A \leq u_A - 2^i$ and $l_B + 2^i < m_B \leq u_B - 2^i$. If these conditions are not satisfied, then the protocol is aborted. Otherwise, if $m_A \geq m_B$, the protocol sets u_A to be m_A and l_B to be m_B. Otherwise it updates l_A to m_A and u_B to m_B. Note that the lower and upper bounds are not revealed to either party.
   C. If $m_A < m_B$, then A removes all elements ranked 2^i or less from S_A, while B removes all elements ranked greater than 2^i from S_B. On the other hand, if $m_A \geq m_B$, then A removes all elements ranked higher than 2^i from S_A, while B removes all elements ranked 2^i or lower from S_B.
5. (Here every party has an input set of size 1.) Party A and party B output the result of a *secure computation* of the minimum value of their respective elements.

   In the malicious case: The secure computation checks that the inputs given in this step are consistent with the inputs given earlier. Specifically, for any item other than item 2^j of the original set of A (respectively B), this means that the value must be equal to u_A (respectively u_B). For item 2^k of step A (respectively B), it is verified that its value is greater than l_A (respectively l_B).

Overhead: Since the value j is at most $\log 2k$ and the number of rounds of communication is $(j+1)$, the total number of rounds of communication is $\log(2k)$. In each round, the protocol performs at most one secure computation, which requires a comparison of $(\log M)$ bit integers. Thus the total communication cost is $O(\log M \cdot \log k)$ times the security parameter.

Proof of Correctness Regardless of security issues, we first have to show that the protocol indeed computes the k^{th}-ranked item. We need to show that (a) The preprocessing performed in Steps 1-3 does not eliminate the k^{th}-ranked value and (b) The $(2^{i+1})^{st}$ value of $S_A^i \cup S_B^i$ is the k^{th}-ranked value in $D_A \cup D_B$ for each $i = j-1, \ldots, 0$ (where S_A^i, S_B^i are the sorted sequences maintained by parties A, B, respectively, during iteration i). These two properties are shown in Lemma 1.

Lemma 1. *In Protocol* FIND-RANKED-ELEMENT, *the* $(2^{i+1})^{st}$*-ranked element of* $S_A \cup S_B$ *in round* i *of Step 4 (i.e., the median) is equal to the* k^{th}*-ranked element in* $D_A \cup D_B$*, for* $i = (j-1), \ldots, 0$.

Proof. Note that in the preprocessing (Step 1) we do not eliminate the k^{th}-ranked element since the k^{th}-ranked element cannot appear in position $(k+1)$ or higher in the sorted version of D_A or D_B. Step 2 ensures that both sequences have size exactly k without affecting the k^{th}-ranked element (since padding is performed at the end of the sequences). And, Step 3 not only ensures that the length of both sequences is a power of 2, but also pads S_A and S_B so that the $(2^j)^{th}$ element of the union of the two sequences is the k^{th}-ranked element of $D_A \cup D_B$. This establishes the Lemma for the case where $i = (j-1)$.

The remaining cases of i follow by induction. We have essentially transformed the original problem to that of computing the median between two sets of equal size 2^{i+1}. Note that neither party actually removes the median of $S_A \cup S_B$: if $m_A < m_B$ then there are $2 \cdot 2^i$ points in S_A and S_B that are larger than m_A and $2 \cdot 2^i$ points in S_A and S_B that are smaller than m_B, thus no point in S_A that is less than or equal to m_A can be the median, nor can any point in S_B greater than m_B. A similar argument follows in the case that $m_A > m_B$. Furthermore, the modifications made to S_A and S_B maintain the median of $S_A \cup S_B$ since at each iteration an equal number of elements are removed from above and below the median (exactly half of the points of each party are removed). The lemma follows.

2.2 Security for the Semi-Honest Case

In the semi-honest case, the security definition in the ideal model is identical to the definition which is based on simulation (which we haven't explicitly described). Thus, it is sufficient to show that, assuming that the number of elements held by each party is public information, party A (and similarly party B), given its own input and the value of the k^{th}-ranked element, can simulate the execution of the protocol in the hybrid model, where the comparisons are done by a trusted party (the proof follows by the composition theorem). We describe the proof detail for the case of party A simulating the execution.

Let x be the k^{th}-ranked element which the protocol is supposed to find. Then, party A simulates the protocol as follows:

Algorithm SIMULATE-FIND-RANK
Input: D_A and x known to A. Public parameter k. All items in $D_A \cup D_B$ are distinct.
Output: Simulation of running the protocol for finding the k^{th}-ranked element in $D_A \cup D_B$.

```
1. Party A initializes $S_A$ to be the sorted sequence of its $k$ smallest elements
   in $D_A$
2. If $|S_A| < k$ then Party A pads $(k - |S_A|)$ values of "+∞" to its sequence
   $S_A$.
3. Let $2^j$ be the smallest power of 2 larger than $k$. Party A pre-pads $S_A$ with
   $(2^j - k)$ values of "-∞".
4. For $i = (j-1), \ldots, 0$ :
   A. A computes the $(2^i)^{th}$ element of $S_A$, $m_A$
   B. If $m_A < x$, then the secure computation is made to output 1, i.e.,
      $m_A < m_B$, else it outputs 0.
   C. If $m_A < x$, then A removes all elements ranked $2^i$ or less from $S_A$.
      On the other hand, if $x \leq m_A$, then A removes all elements ranked
      higher than $2^i$ from $S_A$.
5. The final secure computation outputs 1 if $m_A < x$ and 0 otherwise (in this
   case $m_A = x$ is the median).
```

Lemma 2. *The transcript generated by Algorithm* SIMULATE-FIND-RANK *is the same as the transcript generated by Protocol* FIND-RANKED-ELEMENT. *In addition, the state information that Party A has after each iteration of Step 4, namely (S_A, k), correctly reflects the state of Protocol* FIND-RANKED-ELEMENT *after the same iteration.*

Proof. We prove the lemma by induction on the number of iterations. Assume that the lemma is true at the beginning of an iteration of Step 4, i.e. Algorithm SIMULATE-FIND-RANK has been correctly simulating Protocol FIND-RANKED-ELEMENT and its state correctly reflects the state of Protocol FIND-RANKED-ELEMENT at the beginning of the iteration. We show that $m_A < x$ if and only if $m_A < m_B$. If $m_A < x$ then the number of points in S_A^i smaller than x is at least 2^i. If by way of contradiction $m_B \leq m_A$, then $m_B < x$, implying that the number of points in S_B^i smaller than x is at least 2^i. Thus the total number of points in $S_A^i \cup S_B^i$ smaller than x would be at least 2^{i+1}, contradicting that x is the median. So, $m_A < m_B$. On the other hand, if $m_A < m_B$, and by way of contradiction, $m_A \geq x$, then $x \leq m_A < m_B$. Thus the number of points in S_B^i greater than x is strictly more than 2^i. Also, at least 2^i points in S_A^i are greater than x. Thus, the number of points in $S_A^i \cup S_B^i$ greater than x is strictly more than 2^{i+1}, again contradicting that x is the median. So, $m_A < x$. Thus, the secure computations in Step 4 of Algorithm Simulate-Find-Rank return the same outputs as in Protocol FIND-RANKED-ELEMENT.

Duplicate Items Protocol FIND-RANKED-ELEMENT preserves privacy as long as no two input elements are identical (this restriction must be met for each party's input, and also for the union of the two inputs). The reason for this restriction is that the execution of the protocol reveals to each party the exact number of elements in the other party's input which are smaller than the k^{th} item of the union of the two inputs. If all elements are distinct then given the k^{th}-ranked value each party can compute the number of elements in its own input that are smaller than it, and therefore also the number of such elements in the other party's input. This information is sufficient for simulating the execution of the protocol. However, if the input contains identical elements then given the k^{th}-ranked value it is impossible to compute the exact number of elements in the other party's input which are smaller than it and to simulate the protocol. (For example, if several items in A's input are equal to the k^{th}-ranked element then the protocol could have ended with a comparison involving any one of them. Therefore A does not know which of the possible executions took place.)

Handling duplicate items Protocol FIND-RANKED-ELEMENT-MULTIPARTY in Section 3 can securely computed the k^{th}-ranked item even if the inputs contain duplicate elements, and can be applied to the two-party case (although with $\log M$ rounds, instead of $\log k$). Also, protocol FIND-RANKED-ELEMENT can be applied to inputs that might contain identical elements, if they are transformed into inputs containing distinct elements. This can be done, for example, in the following way: Let the total number of elements in each party's input be n. Add $\lceil \log n \rceil + 1$ bits to every input element, in the least significant positions. For every element in A's input let these bits be a "0" followed by the rank of the element in a sorted list of A's input values. Apply the same procedure to B's inputs using a "1" instead of a "0". Now run the original protocol using the new inputs, but ensure that the output does not include the new least significant bits of the k^{th} item. The protocol is privacy preserving with regard to the new inputs (which are all distinct). Also, this protocol does not reveal to party A more information than running the original protocol with the original inputs and in addition providing A with the number of items in B's input which are smaller than the k^{th} value (the same holds of course w.r.t. B). This property can be verified by observing that if A is given the k^{th}-ranked element of the union of the two inputs, as well as the number of elements in B's input which are smaller than this value, it can simulate the operation of the new protocol with the transformed input elements.

Hiding the Size of the Inputs Assume that the parties wish to hide from each other the size of their inputs. Note that if k is public then the protocol that we described indeed hides the sizes of the inputs, since each party transforms its input to one of size k. This solution in insufficient, though, if k discloses information about the input sizes. For example, if the protocol computes the median, then k is equal to half the sum of the sizes of the two inputs. We next show now how to hide the size of the inputs when the two parties wish to compute the value of the p^{th} percentile, which includes the case of computing the median (which is the 50^{th} percentile).

p^{th}-Percentile The p^{th} percentile is the element with rank $\lceil \frac{p}{100} \cdot (|D_A| + |D_B|) \rceil$. We assume that an upper bound U on the number of elements held by each party is known. Both parties first pad their inputs to get U elements each, in a way that keeps the value of the p^{th} percentile. For this, if a party A needs to add $X = U - |D_A|$ elements to its input, it adds $\frac{p}{100}X$ elements with value $-\infty$ and $\frac{(100-p)}{100}X$ elements with value $+\infty$ (to simplify the exposition, we assume that $\frac{p}{100}X$ is an integer). Party B acts in a similar way. Then the parties engage in a secure computation of the p^{th} percentile, which is the $(\frac{p}{100} \cdot 2U)^{th}$-ranked element of the new inputs, using the protocol we described above.

2.3 Security for the Malicious Case

We assume that the comparison protocol is secure against malicious parties. We then show that although the malicious party can choose its input values adaptively during the execution of the protocol, it could as well have constructed an input apriori and given it to a trusted third party to get the same output. In other words, although the adversary can define the values of its input points depending on whether that input point needs to be compared or not in our protocol, this does not give it any more power. The proof is composed of two parts. First, we show that the functionality provided by the protocol definition provides the required security. Second, we show how to implement this functionality efficiently.

Lemma 3. *For every adversary A' in the real model there is an adversary A'' in the ideal model, such that the outputs generated by A' and A'' are computationally indistinguishable.*

Proof Sketch: Based on the composition theorem, we can consider only a protocol in the hybrid model where we assume that the comparisons are done securely by a trusted party. (We actually need here a composition theorem for a reactive scenario. We refer the reader to [3,5] for a treatment of this issue.)

Visualize the operation of A' as a binary tree. The root is the first comparison it performs in the protocol. Its left child is the comparison which is done if the answer to the first comparison is 0, and the right child is the comparison that happens if the first answer is 1. The tree is constructed recursively following this structure, where every node corresponds to a comparison done at Step 4(B). We add leaves corresponding to the secure computation of Step 5 of the protocol following the sequence of comparisons that lead to a leaf.

Fix the random input used by the adversary A'. We also limit ourselves to adversaries that provide inputs that correspond to the bounds maintained by the protocol (otherwise the protocol aborts as in early termination, and since this is legitimate in the ideal model, we are done). We must generate an input to the trusted party that corresponds to the operation of A'. Let us run A' where we provide it with the output of the comparisons. We go over all execution paths (i.e. paths in the tree) by stopping and rewinding the operation. (This is possible since the tree is of logarithmic depth.) Note that each of the *internal* nodes corresponds to a comparison involving a *different* location in the sorted list that A' is required to generate according to the protocol. Associate with each node the value that A' provides to the corresponding comparison.

Observe the following facts:

– For any three internal nodes L, A, R where L and R are the left and right children of A, the bounds checked by the protocol enforce that the value of L is smaller than that of A, which is smaller than that of R. Furthermore, an inorder traversal of the

internal nodes of the tree results in a list of distinct values appearing in ascending order.
- When the computation reaches a leaf (Step 5), A' provides a single value to a comparison. For the rightmost leaf, the value is larger than any value seen till now, while for each of the remaining leaves, the value is the same as the value on the rightmost internal node on the path from the root to the leaf (this is enforced by checking that the value is the same as u_A or u_B respectively).
- Each item in the input of A' is used in at most a single internal node, and exactly a single leaf of the tree.

Consequently, the values associated with the leaves are sorted, and agree with all the values that A' provides to comparisons in the protocol. We therefore use these values as the input to the trusted third party in the ideal model. When we receive the output from the trusted party we simulate the route that the execution takes in the tree, provide outputs to A' and B, and perform any additional operation that A' might apply to its view in the protocol.[6] □

Implementing the Functionality of the Malicious Case Protocol The functionality that is required for the malicious case consists of using the results of the first i comparisons in order to impose bounds on the possible inputs to the following comparison. This is a *reactive secure computation*, which consists of several steps, where each step operates based on input from the parties and state information that is delivered from the previous step. This scenario, as well as appropriate security definitions and constructions, was described in [3,5]. (We are interested, however, in a simpler synchronous environment with secure channels.)

In order to implement secure reactive computation, each step should output shares of a state-information string, which are then input to the following step. The shares must be encrypted and authenticated by the secure computation, and be verified and decrypted by the secure computation of the following step. This functionality can be generically added to the secure computation

3 Multi-party Computation of the k^{th} Ranked Element

We now describe a protocol that outputs the exact value of the k^{th}-ranked element of the union of multiple databases. For this protocol we assume that the elements of the sets are integer-valued, but they need not be distinct. Let $[\alpha, \beta]$ be the (publicly-known) range of input values, and let $M = \beta - \alpha + 1$. The protocol runs a series of rounds in which it (1) suggests a value for the k^{th}-ranked element, (2) performs a secure computation to which each party reports the number of its inputs which are smaller than this suggested value, adds these numbers and compares the result to k, and (3) updates the guess. The number of rounds of the protocol is logarithmic in M.

[6] Note that we are assuming that the inputs can be arbitrary Real numbers. If, on the other hand, there is some restriction on the form of the inputs, the protocol must verify that A' provides values which are consistent with this restriction. For example, if the inputs are integers then the protocol must verify that the distance between the reported median and the bounds is at least half the number of items in the party's input (otherwise the input items cannot be distinct).

Malicious adversaries. We describe a protocol which is secure against semi-honest adversaries. Again, the protocol can be amended to be secure against malicious adversaries by verifying that the parties are providing it with consistent inputs. We specify in the protocol the additional functionality that should be implemented in order to provide security against malicious adversaries.

Protocol FIND-RANKED-ELEMENT-MULTIPARTY
Input: Party P_i, $1 \leq i \leq s$, has database D_i. The sizes of the databases are public, as is the value k. The range $[\alpha, \beta]$ is also public.
Output: The k^{th}-ranked element in $D_1 \cup \cdots \cup D_s$.

1. Each party ranks its elements in ascending order. Initialize the current range $[a, b]$ to $[\alpha, \beta]$ and set $n = \sum |D_i|$.
   In the malicious case: Set for each party i bounds $(l)^i = 0,\ (g)^i = 0$. These values are used to bound the inputs that party i reports in the protocol. $(l)^i$ reflects the number of inputs of party i strictly smaller than the current range, while $(g)^i$ reflects the number of inputs of party i strictly greater than the current range.
2. Repeat until “done”
   (a) Set $m = \lceil (a + b)/2 \rceil$ and announce it.
   (b) Each party computes the number of elements in its database which are strictly smaller than m, and the number of elements strictly greater than m. Let l_i and g_i be these values for party i.
   (c) The parties engage in the following secure computation:
   In the malicious case: Verify for every party i that $l_i + g_i \leq |D_i|$, $l_i \geq (l)^i$, and $g_i \geq (g)^i$. In addition, if $m = \alpha$, then we check that $l_i = 0$; or if $m = \beta$, we verify that $g_i = 0$.
      – Output “done” if $\sum l_i \leq k - 1$ and $\sum g_i \leq n - k$. (This means that m is the k^{th}-ranked item.)
      – Output “0” if $\sum l_i \geq k$. In this case set $b = m - 1$. (This means that the k^{th}-ranked element is smaller than m.)
      In the malicious case: Set $(g)^i = |D_i| - l_i$. (Since the left endpoint of the range remains the same, $(l)^i$ remains unchanged.)
      – Output “1” if $\sum g_i \geq n - k + 1$. In this case set $a = m + 1$. (This means that the k^{th}-ranked element is larger than m.)
      In the malicious case: Set $(l)^i = |D_i| - g_i$.

Correctness: The correctness of this algorithm follows from observing that if m is the k^{th}-ranked element then the first condition will be met and the algorithm will output it. In the other two cases, the k^{th}-ranked element is in the reduced range that the algorithm retains.

Overhead: The number of rounds is $\log M$. Each round requires a secure multiparty computation that computes two summations and performs two comparisons. The size of the circuit implementing this computation is $O(s \log M)$, which is also the number of input bits. The secure evaluation can be implemented using the protocols of [11,1,8].

Security for the semi-honest case: We provide a sketch of a proof for the security of the protocol. Assume that the multi-party computation in step 2(c) is done by a trusted party. Denote this scenario as the hybrid model. We show that in this case the protocol is secure against an adversary that controls up to $s-1$ of the parties. Now, implement the multi-party computation by a protocol which is secure against an adversary that controls up to t parties, e.g. using [11,1,8]. (Of course, $t < s-1$ in the actual implementation, since the protocols computing the "simple" functionalities used in the hybrid model are not secure against $s-1$ parties, but rather against, say, any coalition of less than $s/3$ corrupt parties.) It follows from the composition theorem that the resulting protocol is secure against this adversary.

In the hybrid model, the adversary can simulate its view of the execution of the protocol, given the output of the protocol (and without even using its input). Indeed, knowing the range $[a, b]$ that is used at the beginning of a round, the adversary can compute the target value m used in that round. If m is the same as the output, it concludes that the protocol must have ended in this round with m as the output (if the real execution did not output m at this stage, m would have been removed from the range and could not have been output). Otherwise, it simply updates the range to that side of m which contains the output (if the real execution had not done the same, the output would have gone out of the active range and could not have been the output). Along with the knowledge of the initial range, this shows that the adversary can simulate the execution of the protocol.

Security for the malicious case: We show that the protocol is secure given a secure implementation of the functionality that is described in Step 3 of algorithm FIND-RANKED-ELEMENT-MULTIPARTY. Since this is a multi-party reactive system we refer the reader to [3,5] for a description of such a secure implementation. (The idea is that the parties run a secure computation of each step using, e.g., the protocol of [1]. The output contains encrypted and authenticated *shares* of the current state, which are then input to the computation of the following step, and checked by it.)

For every adversary that corrupts up to $s-1$ parties in the computation in the hybrid model, there is an adversary with the same power in the ideal model. We limit the analysis to adversaries that provide inputs that agree with all the boundary checks in the algorithm (otherwise the protocol aborts, and this is a legitimate outcome in the ideal model).

Imagine a tree of size M corresponding to the comparisons done in the protocol (i.e. the root being the comparison for $m = (\beta - \alpha)/2$, etc.). Consider also the range $[\alpha, \beta]$ where each element is associated with the single node u in the tree in which m is set to the value of this element. Fix the random values (coin flips) used by the adversary in its operation. Run the adversary, with rewinding, checking the values that are given by each of the parties it controls to each of the comparisons. The values that party i provides to the comparison of node u define, for the corresponding element in the range, the number of items in the input of party i which are smaller than, larger than, and equal to that value.

Assume that we first examine the adversary's behavior for the root node, then for the two children of the root, and continue layer by layer in the tree. Then the boundary checks ensure that the nodes are consistent. Let l_u, e_u, g_u denote the number of items that are specified by the adversary to be less than, equal to, and greater than u, respectively. Then, for any three nodes L, A, R that appear in this order in an inorder traversal of the tree, the boundary checks ensure that $l_L + e_L \leq l_A$ and $g_R + e_R \leq g_A$. Since $l_A + e_A + g_A = l_R + e_R + g_R$, the second inequality implies that $l_A + e_A \leq l_R$. Thus, for any two nodes u and v with $u < v$, we have $l_u + e_u \leq l_v$. In particular, for $i = \alpha, \ldots, \beta - 1$, we have $l_i + e_i \leq l_{i+1}$, which implies that $\Sigma_{i=\alpha}^{\beta} e_i \leq l_\beta + e_\beta - l_\alpha$. Since $l_\alpha = 0$ and $g_\beta = 0$ (enforced by our checks), we know that $l_\beta + e_\beta - l_\alpha = D_i$. Thus, $e_i = l_{i+1} - l_i$ for $\alpha \leq i < \beta$.

We use the result of this examination to define the input that each corrupt party provides to the trusted party in the ideal model. We set the input to contain e_u items of value u, for every $u \in [\alpha, \beta]$. The trusted party computes the k^{th} value (say, using the same algorithms as in the protocol). Since in the protocol itself the values provided by each party depend only on the results of previous comparisons (i.e. path in the tree) the output of the trusted party is the same as in the protocol.

Acknowledgements

We would like to thank Yehuda Lindell and Kobbi Nissim for stimulating discussions on this work.

References

1. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proc. 22nd Annual ACM Symposium on the Theory of Computing*, pages 503–513, 1990.
2. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
3. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.
4. R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, and R. Wright. Selective private function evaluation with applications to private statistics. In *Proc. of 20th ACM Symposium on Principles of Distributed Computing*, pages 293–304, 2001.
5. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two party computation. In *Proc. 34th ACM Symp. on the Theory of Computing*, pages 494–503, 2002.
6. J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure multiparty computation of approximations. In *Proc. 28th ICALP*, pages 927–938, 2001.
7. M. Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *CT-RSA 2001: The Cryptographers' Track at RSA Conference*, pages 457–472, 2001.
8. M. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *Proc. 24th ACM Symp. on the Theory of Computing*, pages 699–710, 1992.
9. P. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proc. 23rd Int. Conf. Very Large Data Bases*, pages 466–475, 1997.
10. O. Goldreich. Secure multi-party computation. In *Theory of Cryptography Library*, 1998.
11. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing*, pages 218–229, 1987.
12. H. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proc. 24th Int. Conf. Very Large Data Bases*, pages 275–286, 1998.

13. E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.
14. Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.
15. M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *Proc. 33rd Annual ACM Symposium on Theory of Computing*, pages 590–599, 2001.
16. V. Poosala, V. Ganti, and Y. Ioannidis. Approximate query answering using histograms. *IEEE Data Engineering Bulletin*, 22(4):5–14, 1999.
17. M. Rodeh. Finding the median distributively. *Journal of Computer and Systems Sciences*, 24:162–166, 1982.
18. A. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.