J. Cryptol. (2010) 23: 373–401
DOI: 10.1007/s00145-010-9059-9

Journal of
CRYPTOLOGY

Secure Computation of the Median (and Other Elements of Specified Ranks)

Gagan Aggarwal[*]
Google Research, Mountain View, CA, USA
gagan@cs.stanford.edu

Nina Mishra[†]
Search Labs, Microsoft Research, Mountain View, CA, USA
ninam@microsoft.com

Benny Pinkas[‡]
Department of Computer Science, University of Haifa, Haifa, Israel
benny@pinkas.net

Communicated by Dwork.

Received 23 January 2007 and revised 29 December 2009
Online publication 5 February 2010

Abstract. We consider the problem of securely computing the kth-ranked element of the union of two or more large, confidential data sets. This is a fundamental question motivated by many practical contexts. For example, two competitive companies may wish to compute the median salary of their combined employee populations without revealing to each other the exact salaries of their employees. While protocols do exist for computing the kth-ranked element, they require time that is at least linear in the sum of the sizes of their combined inputs. This paper investigates two-party and multi-party protocols for both the semi-honest and malicious cases. In the two-party setting, we prove that the problem can be solved in a number of rounds that is logarithmic in k, where each round requires communication and computation cost that is linear in b, the number of bits needed to describe each element of the input data. In the multi-party setting, we prove that the number of rounds is linear in b, where each round has overhead proportional to b multiplied by the number of parties. The multi-party protocol can be used in the two-party case. The overhead introduced by our protocols closely match the communication complexity lower bound. Our protocols can handle a malicious adversary via simple consistency checks.

Key words. Secure function evaluation, Secure multi-party computation, kth-ranked element, Median, Semi-honest adversary, Malicious adversary

1. Introduction

For an ordered set $S \subset \mathbb{F}$, the *kth-ranked element* is the value $x \in S$ that is ranked k when the set S is sorted in increasing order. Of particular interest is the median, which is the element with rank $p = \lceil |S|/2 \rceil$. Given two parties A and B with data sets $D_A, D_B \subset \mathbb{F}$, respectively, we consider the problem of privately computing the kth-ranked element of $D_A \cup D_B$. We also consider this problem in the multi-party case.

Secure computation of the kth-ranked element is a fundamental, primitive operation with many potential applications. Of particular interest are those settings where the individual data sets are very large, contain proprietary information, yet where the kth-ranked element is of mutual interest. For example, consider two health insurance companies wishing 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 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. Typically, academic departments report only a small number of statistics like the minimum, maximum, average and median salary for assistant, associate and full professor positions. The Taulbee survey is thus able to publish only limited aggregate information. A privacy-preserving, multi-party solution for the kth-ranked element would enable universities to quickly compute the median salary without trusting individual salaries to Taulbee. Such a protocol would also facilitate computation of histograms [14,19,29] in a privacy-preserving manner.

1.1. *Prior Work*

The problem we discuss is referred to as *Secure Function Evaluation (SFE)* in the cryptography literature. 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., [16,33]). 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.[1] 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. Let M be the size of the domain $\mathbb{F}$ from which the data

* G. Aggarwal's work done at HP Labs and Stanford University, and supported in part by a Stanford Graduate Fellowship, NSF Grant ITR-0331640 and NSF Grant EIA-0137761.

† N. Mishra's work partially done at HP Labs and the University of Virginia. Research supported in part by NSF grant EIA-013776.

‡ Most of this work was done while B. Pinkas was at HP Labs. Research supported in part by the Israel Science Foundation (grant number 860/06).

[1] The interested reader can find a detailed description of these protocols in the references above. Alternatively, descriptions of the two-party protocols are available at, e.g., [15,24], and descriptions of the multi-party protocols can be found, for example, in [3,13,15].

sets D_A and D_B are drawn, and let $n = |D_A| + |D_B|$ be the total number of the input elements. Then, the kth-ranked element can be computed via a circuit of size $\Omega(n \log M)$ (since reading 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 [27] show that any two-party protocol can be translated into a secure computation protocol. The effect is that a protocol with communication complexity of c bits is transformed to a secure protocol that performs c invocations of oblivious transfer (or SPIR) from a data set of length 2^c. Since there is a protocol, due to Karchmer, for computing the median with $\log n$ communication [21], the implication is that the size of the data set for the OT/SPIR invocations is polynomial in n, and the communication is $\log n$ times that of the OT/SPIR protocol. If the protocol uses SPIR based on the PIR protocol of Cachin et al. [6], it obtains polylogarithmic communication overhead. The drawback of the protocol, in addition to hidden constants, is that it requires application of a public-key operation to each item in the data set, i.e., a number of times that is polynomial in n.

1.2. *Results*

The results in [16,33] and [27] are quite powerful in that they enable general transformations from known algorithms to secure algorithms. Our interest, however, is to determine how efficiently a specific function, namely the kth-ranked element, can be computed. We are motivated by applications where the total number of data points (or values) owned by the parties (n) is very large, and thus even a linear communication and computation overhead might be prohibitive. (Even taking results on extending oblivious transfers [18] into account, the overhead is $\Omega(n)$). Thus, we describe protocols with sublinear communication and computation overhead. Specifically, in the two-party case, we reduce the computation of the kth-ranked element to $O(\log k)$ secure comparisons of $(\log M)$-bit inputs,[2] 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 kth-ranked element to $O(\log M)$ simple secure computations that involve additions and a comparison of $(\log M)$-bit long numbers. This protocol, too, can be made to be secure even if all but a single party are malicious adversaries. Interestingly, the multi-party solution can be applied to the two-party scenario if it uses secure two-party protocols as primitives. The multi-party protocol can even be applied directly to inputs that contain duplicate items, whereas the two-party protocol requires inputs comprising of distinct inputs.

The advantage of our two-party solution is that the number of rounds is logarithmic in the number of input items, whereas the number of rounds of the multi-party solution is logarithmic in the size of the domain of possible input values. We note that the communication complexity lower bound for computing the median is $\min\{\log n, \log M\}$ [21] whereas our result entails a communication cost of $O(\log n \cdot \log M)$ for a secure computation.

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

1.3. *Techniques*

The protocols given here are modifications of well-known algorithms in the communication complexity literature [21,30]. Our contribution is the modifications and proofs of security that result in privacy-preserving solutions, for both semi-honest and malicious adversaries. For the semi-honest case, our proofs introduce a *binary search simulation technique* where we demonstrate that a sequence of binary search decisions can be simulated from the final output of the protocol. (This technique was later used by other researchers, see, e.g., [1].)

In the two-party case, the algorithm from the communication complexity literature requires that each party repeatedly determine whose median is larger. Based on this information, each party eliminates half of its input values and repeats the computation of whose median is larger on the remainder of its input values. Our binary search simulation proof demonstrates that this sequence of answers can be determined from the final median of the combined data sets.

In the multi-party setting, a binary search is performed over a range of input values $[a, b]$. Each party reports the number of values it possesses below and above the midpoint value $\frac{a+b}{2}$. Based on the answers provided by all the parties, the range is cut in half. Our binary search simulation proof demonstrates that this sequence of decisions of how to cut the interval in half can be simulated from the output, i.e., from the final median of the combined data sets.

For the malicious case, our proofs are surprisingly simple. We describe *consistency tests* that suffice for weeding out a malicious adversary. (This technique was later used by other researchers, see, e.g., [31] Sect. 4.) In the two-party case, the consistency test ensures that the median value repeatedly reported is appropriately lower and upper bounded by previous median values reported. If this consistency check is satisfied then we prove that there exists an input the party could have provided to a trusted third party in the ideal model.

In the multi-party case, the test confirms that the number of values repeatedly reported to be less than or greater than a value is consistent with answers the party has reported to be less than and greater than midpoint values of previous intervals. If this consistency check is satisfied, then we prove that there exists an input the party could have provided to a third party.

1.4. *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., [9,11,24]), and reduce this task to many invocations of secure computation of simpler functions of small inputs (but unlike these constructions, we also design protocols that are secure against malicious adversaries). That is, we describe a protocol for computing the kth-ranked value that uses oracle queries to a few simple functionalities and is secure if these functionalities are computed by a trusted oracle. A composition theorem (see [7,8] and discussions below) shows that if the oracle queries are replaced by secure protocols, then the resulting combined protocol is also secure. The result of the reduction is a distributed protocol whose overhead is sublinear in the size of the inputs and is actually feasible even for very large inputs. We also note that the protocol computes the *exact* value of the kth-ranked item, rather than computing an approximation as in [11].

1.5. *Overview*

The rest of the paper is organized as follows. In Sect. 2, we give standard definitions of semi-honest vs. malicious adversaries and security via the real and ideal model. In Sect. 3, a protocol for efficient, two-party computation of the kth-ranked element is given. Finally, in Sect. 4, a multi-party protocol is described for computing the kth-ranked element.

2. Preliminaries

Our security definitions are based on a comparison to an ideal setting where there is a trusted third party (TTP), each party provides its private input to the TTP and the TTP outputs the appropriate function of the combined data. In reality, since no such ideal setting exists, we describe a real model, where parties can only communicate with each other. The communication is said to be secure if the parties learn no more than they would have in the ideal setting with a TTP. The security definitions that we use, for both the two-party and the multi-party scenarios, are based on [7,15]. For a detailed discussion of security definitions, we refer the reader to [7,15].

We begin by defining an adversary for both the semi-honest and malicious case. Next, we formally define security for the two-party and multi-party setting. Finally, we describe a composition theorem that we will invoke in many of our proofs.

2.1. *Semi-Honest vs. Malicious Adversary*

In order to prove that our protocols are secure, it is important that we define what our adversaries can and cannot do. We consider two typical kinds of adversaries studied in the literature: semi-honest and malicious.

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 make deductions about the inputs of the other parties. Despite this behavior, the goal is to ensure that such an adversary following the protocol still cannot learn more than it would have had it communicated its input to a trusted third party and received the final output.

A *malicious adversary* can be viewed as a fixed, but arbitrary program that controls a subset of parties by dictating their behavior during the execution of the protocol. A malicious adversary may not follow the rules of the protocol. There are, however, several things that a malicious adversary can do that 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. Regarding (1) and (3), other parties may be able to detect that one of their peers is not participating in the protocol, and in many scenarios these parties can then take measures against the corrupt party (this is different than other types of malicious activity which are not easily detected). This property might deter parties from behaving in this way. Regarding (3), which affects the fairness of the protocol, there is no perfect solution for this issue and existing solutions are quite complex. Following [7,15] we do not consider solutions for the fairness of the protocol. Regarding (2), a malicious adversary could anyway provide a substituted value to the TTP—thus since our security goal is to be as private as a TTP, security will be equivalent to the idealized setting.

2.2. *Security in the Two-Party Case*

In the security definitions that follow, 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 the definitions to the case where all parties compute the same function f. To simplify the definitions we do not explicitly handle auxiliary inputs (which are covered by the definitions in [7,15]).

Prior to defining security, we define some terms and notation that will be useful later. We say that a function q is *negligible* if for every $c > 0$ and for all sufficiently large n, $q(n) < 1/n^c$. Two sequences of distributions $\{C_n\}_{n\in N}$ and $\{D_n\}_{n\in N}$ are said to be *computationally indistinguishable* if for any probabilistic polynomial time (PPT) algorithm A, $|\Pr(A(C_n) = 1) - \Pr(A(D_n) = 1)|$ is negligible in n. We use the notation $\mathsf{IDEAL}_{A,B}(X, Y) = (\phi, \rho)$ to mean that two parties A and B, possessing inputs X and Y, respectively, receive outputs ϕ and ρ, respectively.

Definition 1 (The Ideal Model, Two-Party Case). 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).
- Similarly, if A is honest
  - $\mathsf{IDEAL}_{A,B}(X, Y)$ equals $(f(X, Y'), B_O(Y, r, f(X, Y')))$, where $Y' = B_I(Y, r)$ (in the case that B did not abort the protocol),
  - or, $\mathsf{IDEAL}_{A,B}(X, Y)$ equals $(-, B_O(Y, r, f(X, Y')))$, where $Y' = B_I(Y, r)$ (if B terminated the protocol prematurely).

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, Two-Party Case, Semi-Honest & Malicious). 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, including changing its input or sending messages inconsistent with Π.

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, Two-Party Case, Semi-Honest & Malicious). 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 $\mathsf{IDEAL}_{A,B}(X,Y)$.

We note that in the semi-honest case the security definition given here is identical to a definition which is based on simulation (which we have not explicitly described; see definition and equivalence theorem in [15]). Thus, it is sufficient to show that party A (and similarly party B), given its own input and output, can simulate the execution of the protocol. In other words, it is sufficient to show that given its input and output alone, party A can generate a string whose distribution is similar to the distribution of the messages exchanged during the protocol. Intuitively, this shows that everything that A learns in the protocol it can compute just as well from its input and from the function computed by the protocol. Therefore, the execution of the protocol reveals nothing but its desired output. We use this simulation based definition to prove below the security of our protocol in the semi-honest setting.

2.3. *Security in the Multi-Party Case*

We now provide security definitions for the multi-party case. Assume that there are n parties. We denote by J the set of parties under the control of the malicious adversary. Note that J is not known to all parties. We have that $J = \{j_1, \dots, j_t\} \subseteq \{1, \dots, n\}$ and $\bar{J} = \{1, \dots, n\} \backslash J$ and $(x_1, \dots, x_n)_J = (x_{j_1}, \dots, x_{j_t})$. $f_J(\bar{x})$ is defined to be a vector with $|J|$ entries, all of which are equal to $f(\bar{x})$.

Similar to the two-party case, a strategy for the adversary A in the ideal model is a pair of PPT algorithms: $A_I(\bar{x}_J, J, r)$ that uses the input of all corrupt parties, their identities and a sequence of coin flips r to generate the input that the corrupt parties send to the trusted party, and $A_O(\bar{x}_J, J, r, f_J)$ which takes as an additional input the output f_J, namely the outputs that the corrupt parties receive from the TTP, and generates the final outputs of the corrupt parties.

Definition 4 (The Ideal Model, Multi-Party Case). An adversary is represented by a set J of players that it controls, and a probabilistic polynomial time strategy A for controlling these players. The strategy is limited to controlling the input that the members of J provide to the trusted party, the modifications that they may apply to the output they receive from it, and to deciding whether to abort the protocol. We assume that Party 1 is the first to receive the output from the trusted party, and therefore if the adversary controls this party it can learn the output while preventing legitimate parties from learning it.

Denote the input as $\bar{x} = (x_1, \ldots, x_n)$. The execution in the ideal model, denoted $\mathsf{IDEAL}_{f,J,A}(\bar{x})$, is defined to be

- If Party 1 is honest ($1 \notin J$),

$$\mathsf{IDEAL}_{f,J,A}(\bar{x}) = \big(f_{\bar{J}}(\bar{x}'), A_O\big(\bar{x}_J, J, r, f_J(\bar{x}')\big)\big)$$

  where r is the randomness used by the adversary, and $\bar{x}' = (x_1', \ldots, x_n')$ such that $x_j' = x_j$ for $j \notin J$, and $\bar{x}_J' = A_I(\bar{x}_J, J, r)$. Namely, the adversary can change the input of the parties it controls from $\bar{x}_J$ to $\bar{x}_J' = A_I(\bar{x}_J, J, r)$. The trusted party then computes the output $f(\bar{x}')$, and the adversary can change the output of the parties it controls to $A_O(\bar{x}_J, J, r, f_J(\bar{x}'))$.
- If Party 1 is controlled by the adversary ($1 \in J$) then

$$\mathsf{IDEAL}_{f,J,A}(\bar{x}) = \big(\perp^{|\bar{J}|}, A_O\big(\bar{x}_J, J, r, f_J(\bar{x}')\big)\big)$$

  if the adversary decides to abort the protocol after Party 1 receives its output, or, if it decides to continue with the protocol

$$\mathsf{IDEAL}_{f,J,A}(\bar{x}) = \big(f_{\bar{J}}(\bar{x}'), A_O\big(\bar{x}_J, J, r, f_J(\bar{x}')\big)\big).$$

Definition 5 (The Real Model, Multi-Party Case, Semi-Honest & Malicious). Let f be as in Definition 4, Π be an n-party protocol for computing f, and (J, A) define the operation of the adversary. In the *semi-honest case,* A could define arbitrary input and output functions for the parties in J. In the *malicious case*, A can define arbitrary behavior for the parties in J throughout the protocol.

The joint execution of Π under (J, A) in the real model, denoted $\mathsf{REAL}_{\Pi,J,A}(\bar{x})$, is defined as the output sequence resulting from the interaction between the n parties.

Definition 6 (Security, Multi-Party Case, Semi-Honest & Malicious). Let f and Π be as in Definition 5. Protocol Π **securely computes** f if for every PPT adversary A in the real model (of Definition 5) there is a PPT algorithm B in the ideal model (of Definition 4), such that for every $J \subseteq \{1, \ldots, m\}$, $\mathsf{REAL}_{\Pi,J,A}(\bar{x})$ is computationally indistinguishable from $\mathsf{IDEAL}_{f,J,B}(\bar{x})$.

We note that as in the two-party case, security in the semi-honest setting is equivalent to security according to a simulation based definition, see [15] for details.

2.4. *A Composition Theorem*

Our protocols implement the computation of the kth-ranked element by running many invocations of secure computation of simpler functionalities. The security of these constructions is proved using the secure composition theorem of Canetti [7,15] (which builds upon the previous work of [2,16,17,26]). Loosely speaking, the theorem enables us to construct protocols in a *hybrid model* where a protocol for a function F uses a trusted party which securely computes simpler functionalities $f_1, \dots, f_\ell$. The secure composition theorem states that if we consider security in terms of comparing the real computation to the ideal model, then we can take a protocol which is secure in the hybrid model, replace the calls it makes to the trusted party by calls to secure protocols computing $f_1, \dots, f_\ell$, and obtain a protocol which is secure. We state the theorem below for the case of multi-party protocols (the two-party case is identical).

Theorem 1 (Secure Composition, Based on Corollary 7 of [7]). *Let $t < n$ and let $f_1, \dots, f_\ell$ be n-party functions. Let π be an n-party protocol* (*operating in the hybrid mode*) *which makes calls to a trusted party computing $f_1, \dots, f_\ell$ and is secure* (*in the sense of Definition 6*) *against adversaries which control up to t parties. Let $\rho_1, \dots, \rho_\ell$ be n-party protocols for computing $f_1, \dots, f_\ell$, respectively, which are secure against adversaries which control up to t parties. If $\pi^{\rho_1,\dots,\rho_\ell}$ is a protocol in which the calls to the trusted party for computing f_i $(1 \le i \le \ell)$ are replaced by an execution of ρ_i, then $\pi^{\rho_1,\dots,\rho_\ell}$ is secure against adversaries that control up to t parties.*

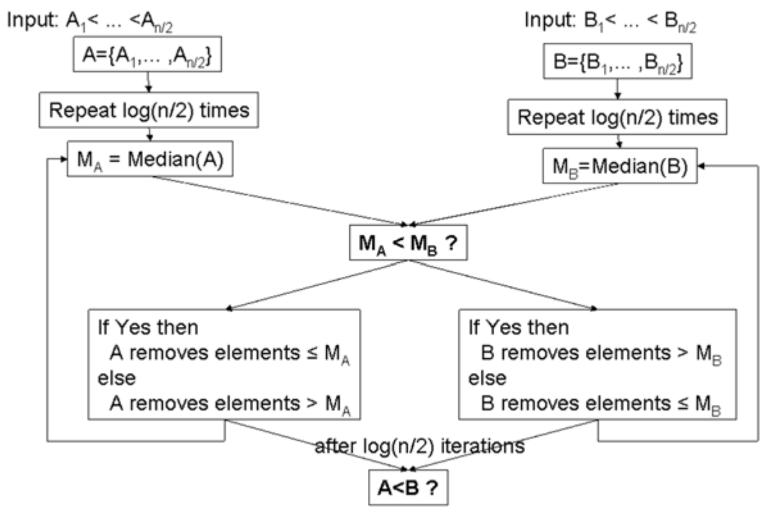
3. Two-Party Computation of the kth Element

This section describes protocols for secure two-party computation of the kth-ranked element of the union of two data sets. The protocols are based on the observation that a natural algorithm for computing the kth-ranked element discloses very little information that cannot be computed from the value of the kth-ranked element itself. Some modification to that protocol can further limit the information that is leaked by the execution, to be equal to the 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 having 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 [21,30] and is depicted in Figs. 1 and 2.[3]

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 remaining 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 from 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 remaining input sets 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

[3] Another variant of the algorithm that is presented there, and is due to Karchmer, reduces the communication overhead to $O(\log n)$ bits (instead of $O(\log^2 n)$). Our protocols do not use this improvement. In any case, the communication associated with the secure computation overshadows the communication overhead of the basic protocol.



Input: $A_1 < \ldots < A_{n/2}$
$A=\{A_1,\ldots,A_{n/2}\}$
Repeat log(n/2) times
M_A = Median(A)
Input: $B_1 < \ldots < B_{n/2}$
$B=\{B_1,\ldots,B_{n/2}\}$
Repeat log(n/2) times
M_B=Median(B)
$M_A < M_B$?
If Yes then A removes elements ≤ M_A else A removes elements > M_A
If Yes then B removes elements > M_B else B removes elements ≤ M_B
after log(n/2) iterations
A<B ?

Fig. 1. A two-party protocol for computing the median.

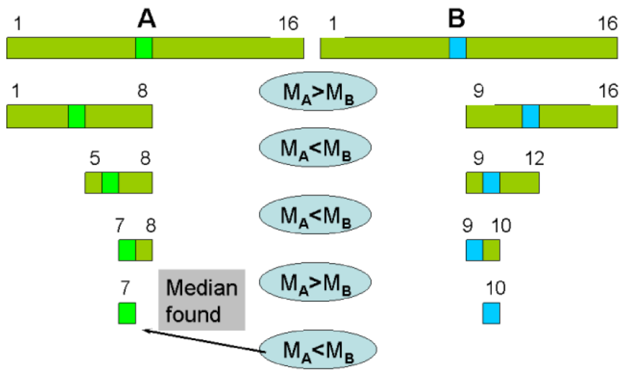


Fig. 2. An illustration of the protocol of Fig. 1 for the case where each party has 16 input elements. Time proceeds vertically.

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., the parties only learn which party's median value is greater, and do not learn any other information about each other's median value. We show below that in this case the protocol is secure. Intuitively, this is true because each party can deduce the result of the comparison from the value of the overall median and its own input. For example, 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, both parties can simulate the results of the comparisons. Consequently, we have a reduction from the problem of securely computing the median of the union to the problem of 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. For ℓ bit inputs, the overhead of the semi-honest version of this secure protocol is ℓ oblivious transfers, which are implemented using $O(\ell)$ modular exponentiations, and $O(\ell)$ applications of a pseudo-random function. The communication is $O(\ell)$ times the length of the security parameter. In the malicious behavior case, we require a more complex comparison protocol that needs to perform additional checks and pass a state from one secure computation to the next. That protocol, described in detail in Sect. 3.1, can be achieved using a generic compiler described in [15,16], or by a compiler which is based on the cut-and-choose methodology [25].

More efficient, non-interactive comparison protocols also exist. For example, Fischlin describes a protocol which uses $O(\ell\lambda)$ modular *multiplications* and has error probability $2^{-\lambda}$ [12], and additional results [4,22,23] that remove the dependency on the error parameter λ. (There are other specialized protocols for this problem. In particular, Cachin suggested a protocol that ensures fairness given a semi-trusted third party [5].)

3.1. *A Protocol for Semi-Honest and Malicious Parties*

We next describe a protocol that finds the kth-ranked element in the union of two data sets 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 kth-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 must also verify 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 also specify the *additional functionality* that is required in order to ensure security against malicious parties. Then in Sect. 3.4 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 secure only against semi-honest adversaries, one should ignore the additional highlighted steps that provide security in the malicious case.

Protocol 1 FIND-RANKED-ELEMENT

```
Input: $D_A$ known only to $A$, and $D_B$ known only 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 "$+\infty$" to its sequence $S_A$. Party $B$ does the same: if $|S_B| < k$ then it pads $(k - |S_B|)$ values of "$+\infty$" 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 "-$\infty$" and Party $B$ pads $S_B$ with $(2^j - k)$ values of "$+\infty$". (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 $i = (j-1), \ldots, 0$ do
5:     $A$ 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.)
6:     $A$ and $B$ engage in a secure computation that 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 $l_A < m_A < u_A$ and $l_B < m_B < u_B$. If these conditions are not satisfied, then the protocol is aborted. Otherwise, if $m_A \geq m_B$, the protocol sets $u_A = m_A$ and $l_B = m_B$. 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. ]
7:     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 $S_B$.
8: end for
9: (By now, both $S_A$ and $S_B$ are of size 1.) Party $A$ and $B$ output the result of a secure computation which computes the smaller of the two remaining 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 $A$ (resp., $B$), this means that the value must be equal to $u_A$ (resp., $u_B$). For the item ranked $2^j$ in the original set of party $A$ (resp., $B$), it is verified that its value is greater than $l_A$ (resp., $l_B$).]
```

Overhead The total number of rounds of communication is $\log(2k)$, since the value j is less than $\log 2k$ and the number of rounds of communication is $(j+1)$. In each round, the protocol performs at most one secure comparison of $(\log M)$-bit integers. A circuit for performing the comparison has $O(\log M)$ gates and $\log M$ inputs. The overhead of a protocol for computing this circuit, secure against semi-honest adversaries, is $\log M$ oblivious transfers (which translate into $O(\log M)$ public key operations, such as exponentiations), $O(\log M)$ symmetric operations, and communication of $O(\log M)$ bits times a security parameter. A protocol secure against malicious adversaries can be run with $O(\log M)$ public key operations and communication dominated by sending $O(s^2 \log M)$ commitments (where s is a statistical security parameter), using the protocol in [25], or with $O(\log M)$ public key operations, and communication of $O(\log M)$ group elements, albeit with much larger constants, based on the protocol of [20]. Thus the total communication cost is $O(\log M \cdot \log k)$ times a security parameter, and the total computational overhead is $O(\log M \cdot \log k)$ exponentiations.

Padding The protocol requires the parties to pad their inputs with the values "$+\infty$" and "$-\infty$". In the malicious case, we must take into account a malicious party that might replace some of its original inputs with, e.g., the "$+\infty$" value, or add a wrong number of these values. To handle this case we define "$+\infty$" and "$-\infty$" to be the maximum and minimum values of the input domain, respectively. As a result, any padded input corresponds to a legitimate input that a malicious party could have sent to the trusted party, where both inputs affect the execution identically.

Proof of Correctness

Regardless of security issues, we first have to show that the protocol indeed computes the kth-ranked item. We need to show that (a) The preprocessing performed in Steps 1–3 does not eliminate the kth-ranked value and (b) The (2^{i+1})st value of $S_A^i \cup S_B^i$ is the kth-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 round i). These two properties are shown in Lemma 1.

Lemma 1. *Let D_A and D_B be data sets held by party A and B, respectively. 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 kth-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 kth-ranked element since the kth-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 kth-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 kth-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. □

3.2. *Security for the Semi-Honest Case*

In the semi-honest case, the security definition which compares the ideal model to the real execution is identical to the definition which is based on simulation (which we have not explicitly described; see definition and equivalence theorem in [15]). 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 kth-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). In other words, party A can generate an output whose distribution is similar to the distribution of the messages exchanged in the protocol. We describe the proof in detail for the case of party A simulating the execution. Let x be the kth-ranked element which the protocol is supposed to find. Then, party A simulates the protocol using the procedure described in Algorithm 2.

Algorithm 2 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 "$+\infty$" 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 "$-\infty$".
4: for $i = (j-1), \ldots, 0$ : do
5:    A computes the $(2^i)$th element of $S_A$, $m_A$.
6:    If $m_A < x$, then the secure computation is made to output 1, i.e., $m_A < m_B$, else it outputs 0.
7:    If $m_A < x$, then $A$ removes all elements ranked $2^i$ or less from $S_A$. On the other hand, if $x \le m_A$, then $A$ removes all elements ranked higher than $2^i$ from $S_A$.
8: end for
9: 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. □

Theorem 2. *Protocol* FIND-RANKED-ELEMENT *securely computes the kth-ranked element of the union of inputs of A and B, for the case of semi-honest adversaries, assuming all inputs of A and B are distinct.*

Proof. The proof follows from Theorem 1 (composition theorem in the hybrid model), Lemmas 1 and 2. □

3.3. *Variants*

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 kth item of the union of the two inputs. If all elements are distinct then given the kth-ranked value, each party can compute the number of elements in its own input that are smaller than it, and therefore each party can also compute 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 kth-ranked value, it is impossible to compute the exact number of elements in the other party's input which are smaller than it, thus preventing one from simulating the protocol. (For example, if several items in A's input are equal to the kth-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 In order to apply Protocol FIND-RANKED-ELEMENT to inputs that might contain identical elements, the inputs are first transformed to contain only 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". Denote the resulting values as the "distinct representations" of the original values. Now run the original protocol using the new inputs, the distinct representations. The parties learn the kth-ranked element of the new list, which we denote as k_d^{th}. It is the distinct representation of the kth-ranked element of the original values (which we denote as k_o^{th}). The protocol is privacy-preserving with regard to the new inputs (which are all distinct). Also, note that the output of the protocol, k_d^{th}, can be computed by A from k_o^{th} and from knowledge of the number of items in B's input which are smaller than the k_o^{th} value (a similar statement holds for B as well). This property can be verified by observing that if A is given k_o^{th}, 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.

We also note that protocol FIND-RANKED-ELEMENT-MULTIPARTY presented in Sect. 4 can securely compute the kth-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 $\log k$, where M is the size of the input space).

Hiding the Size of the Inputs Consider the case where the two parties wish to hide from each other the size of their inputs. Note that if the value 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 is 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 how two parties can compute the value of the element with rank $\lfloor \phi n \rfloor$, where $0 < \phi < 1$ without revealing "too much" about the size of the inputs. More precisely, let $\phi = \phi_n / \phi_d$, where both ϕ_n and ϕ_d are integers. Then our protocol requires each party to reveal the remainder left when the size of its input is divided by ϕ_d. We note that for small values of ϕ_d, such a revelation is usually acceptable even in cases where the two parties want to hide the size of their respective data sets from each other.

Let U, a multiple of ϕ_d, be a public upper bound on the number of elements held by each party. In the protocol, the two parties pad their inputs with roughly $\phi(2U - |S_A| - |S_B|)$ elements with value $-\infty$ and roughly $(1-\phi)(2U - |S_A| - |S_B|)$ elements with value $+\infty$, and then the parties run a secure computation of the $(2\phi U)$th-ranked element of the new inputs, using the protocol described above. The exact number of elements added by each of the parties depends on the remainder left on dividing $|S_A|$ and $|S_B|$ by ϕ_d as described next.

If both $|S_A|$ and $|S_B|$ are divisible by ϕ_d, then party A pads its input with $\phi(U - |S_A|)$ elements with value $-\infty$, and $(1-\phi)(U - |S_A|)$ elements with value $+\infty$; similarly, party B pads its input with $\phi(U - |S_B|)$ elements with value $-\infty$, and $(1-\phi)(U - |S_B|)$ elements with value $+\infty$. The union of the two inputs now contains $2U$ elements, where the $(2\phi U)$th-ranked element is the (ϕn)th-ranked element of the original inputs.

Otherwise, let $r_A = \phi_d - (|S_A| \bmod \phi_d)$ and let $r_B = \phi_d - (|S_B| \bmod \phi_d)$. The parties share the values of r_A and r_B with each other. First, party A adds $\phi(U - (|S_A| + r_A))$ elements with value $-\infty$ and $(1-\phi)(U - (|S_A| + r_A))$ elements with value $+\infty$;

similarly, party B adds $\phi(U-(|S_B|+r_B))$ elements with value $-\infty$ and $(1-\phi)(U-(|S_B|+r_B))$ elements with value $+\infty$. At this point, party A has $U-r_A$ elements and party B has $U-r_B$ elements. Next, assume without loss of generality that $r_A \geq r_B$ (otherwise, interchange the role of A and B in the following). If $\phi \leq 0.5$ then party A adds $\lceil \phi(r_A+r_B) \rceil \leq r_A$ more elements with value $-\infty$ and adds $r_A - \lceil \phi(r_A+r_B) \rceil$ more elements with value $+\infty$, while party B adds r_B more elements with value $+\infty$. Thus, the element with rank $\lfloor \phi n \rfloor$ in the original inputs becomes the $(2\phi U)$th-ranked element of the new inputs. If $\phi > 0.5$, then party A adds $\lfloor (1-\phi)(r_A+r_B) \rfloor \leq r_A$ more elements with value ∞ and adds $r_A - \lfloor (1-\phi)(r_A+r_B) \rfloor$ more elements with value $-\infty$, while party B adds r_B more elements with value $-\infty$. Again, the union of the two inputs now contains $2U$ elements, where the $(2\phi U)$th-ranked element is the (ϕn)th-ranked element of the original inputs.

Let us now analyze the operation of this modified protocol in the case of malicious adversaries. First, in the case that both $|S_A|$ and $|S_B|$ are divisible by ϕ_d, each party must independently pad its input with "$+\infty$" and "$-\infty$" values. Recall that these values correspond to the maximum and minimum values in the input domain, respectively. The proof of security against malicious adversaries, given in Theorem 3 of Sect. 3.4, shows that a corrupt party's behavior in the real execution corresponds to its operation in the ideal protocol using some legitimate input. In the modified protocol which hides the size of the inputs, this means that the real execution corresponds to some padded input. This padded value corresponds to one or more legitimate original inputs (generated from the padded input by removing the right fraction of "$+\infty$" and "$-\infty$" values). These original inputs could have been sent to the trusted party in the ideal model, and resulted in the same effect as the actual execution in the real model. The situation is more complicated if either $|S_A|$ or $|S_B|$ is not divisible by ϕ_d. In that case, the parties exchange their r_A and r_B values, and have to pad their inputs based on these values. It might be, however, that the padded inputs in the ideal scenario, as are defined in the proof of Theorem 3, do not correspond to these r_A and r_B values. For example, suppose that party B reports a value r_B such that $0 < r_B < r_A$, and it also holds that $\phi \leq 0.5$. We still might find out that B's padded input in the ideal model contains no "$+\infty$" values, although the protocol adds such values to the padded version of B's input. We therefore conclude that if $|S_A|$ and $|S_B|$ are both divisible by ϕ_d then the modified protocol is secure against malicious adversaries, whereas otherwise it is not, unless additional measures are taken to verify that the parties perform the padding according to the protocol.

3.4. *Security for the Malicious Case*

We assume that the comparison protocol is secure against malicious parties in the sense of Definition 3. We then show that although a malicious party can choose its input values adaptively during the execution of the protocol, it could as well have constructed an input a priori 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 which points need to be compared in our protocol, this does not give the adversary any more power. The proof shows that the functionality provided by protocol Find-Ranked-Element provides the required security. Then, we show how to implement this functionality efficiently.

Theorem 3. *Protocol* FIND-RANKED-ELEMENT *securely computes the kth-ranked element of the union of inputs of A and B, for the case of a malicious adversary.*

Proof. Following Definition 3, the proof shows that 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. Based on the composition theorem, we can consider only protocols in the hybrid model where we assume that the comparisons are done securely by a trusted party.

The operation of A' in the protocol can be visualized as a binary tree. An example when each party has eight input values is given in Fig. 3. The root of the tree is the input of A' to the first comparison performed in the protocol. The left child of the root is its input to the second comparison if the answer to the first comparison is "yes", and the right child is its input to the second comparison if the first answer is "no". The tree is constructed recursively following this structure, where every node corresponds to the input provided by A' to a comparison done at Step 6. The leaves of the tree correspond to the input provided by A' to the secure computation of Step 9 of the protocol. Note that there is a unique leaf corresponding to each possible outcome of the sequence of comparisons of Step 6 (one such path to a leaf is highlighted in Fig. 3).

The operation of A' in an actual execution follows a path from the root to a node of this tree. In order to construct a computationally indistinguishable adversary A'' in the ideal model, we assume that the random input used by A' is fixed and known to A''. We also limit ourselves to adversaries A' that provide inputs that correspond to the bounds l_A, u_A maintained by the protocol (otherwise the protocol aborts as in early termination, and since this can be achieved by providing invalid input in the ideal model, the theorem is proved). The adversary A'' needs to give an input to the trusted party in the ideal model such that it can generate a transcript that is computationally indistinguishable from A'. Since we work in the hybrid model, A''s transcript includes only the inputs and outputs of the secure computations of Step 6 and 9 of the protocol and the final output of the protocol.

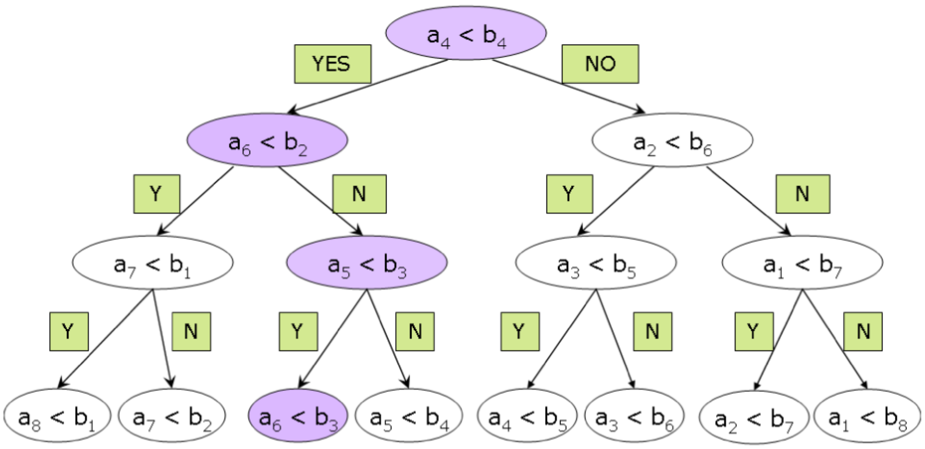


Fig. 3. A depiction of the operation of A' in the proof of Theorem 3.

We describe here a procedure which determines the input that A'' needs to give to the trusted party. The procedure has to learn the entire input of A'. Recall that when running this procedure we do not have access to A''s input and can learn about it only by running it. So we run A', providing it with the output of the comparisons. We can also store the state of A' at a certain point of its execution, and later on restore that state ("rewind" A') and continue the execution from the restored state. We go over all execution paths by stopping and rewinding the operation. Namely, we run A', obtain its input, m_A, to the first comparison, and provide it with the answer that $m_A \geq m_B$ in the first comparison. We obtain the input of A' to the second comparison, provide it again with an answer that $m_A \geq m_B$, and so on. After simulating in this way all j comparisons, we obtain the input that A' provides to the comparison of Step 9. We then rewind A' to the beginning of the last comparison, and in this time we provide it with an answer that $m_A < m_B$. In a similar manner we go over all execution paths of the protocol, which can be viewed as a tree of depth $j+1$ (j comparisons of Step 6 and the final comparison of Step 9 which corresponds to the leaves of the tree). The traversal of this tree is possible in polynomial time since its depth is $j = \log k$. Note that each of the *internal* nodes corresponds to a comparison involving a *different* location in the sorted list that A' is supposed to have as its input. 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 9), 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 lowermost internal node of the path from the root to the leaf, for which the comparison result was that $m_A \geq m_B$ (this property is enforced by the protocol by checking that the value input by A' to Step 9 is the same as u_A).
- 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 can therefore use these values as the input of A'' to the trusted third party in the ideal model. When A'' receives the output from the trusted party, it simulates the route that the execution takes in the tree, and performs any additional operation that A' might apply to its view in the protocol. $\square$

Note that the proof assumes that the inputs are 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 must be integer numbers 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). Namely, Step 6 must check $l_A + 2^i < m_A \leq u_A - 2^i$ and $l_B + 2^i < m_B \leq u_B - 2^i$.

3.4.1. *Implementing the Functionality of the Malicious Case Protocol, and Relation to Reactive Computation*

The protocol for the malicious case runs in consecutive steps, which use the results of the first i comparisons in order to impose bounds on the possible inputs to the following comparison. This type of protocol is similar to the notion of reactive computation [8,10, 28][4] and can be securely implemented using secure protocols for reactive computation. The computation of the kth ranked element does not require, however, a reactive computation, since the parties know their entire inputs at the onset of the protocol. The protocol described in our work consists of several steps, each of which requires the parties to enter additional inputs, but this structure is merely used in order to improve efficiency. If we did not care about efficiency, we could have computed the kth ranked element using a single run of a protocol (say, based on Yao's generic protocol and running in $O(1)$ rounds) to which each party provides its k inputs.

One way of implementing our protocol is to use full pledged reactive computation, consisting 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 [8,10]. (But, unlike the definitions of [8,10] we are only interested in a simpler synchronous environment with secure channels and assume that in each step *all* parties provide an input.) We also describe in Appendix 4.1 a simple modification of Yao's two-party protocol that can be used to implement the protocol.

4. Multi-Party Computation of the kth Ranked Element

We now describe a protocol, Protocol 3, that outputs the value of the kth-ranked element of the union of *multiple* private data sets. 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 kth-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.

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.

[4] 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. General constructions for reactive computation were discussed in [8,10]. In particular, they enable parties to abort the protocol at arbitrary stages.

Protocol 3 FIND-RANKED-ELEMENT-MULTIPARTY

```
Input: Party $P_i$, $1 \le i \le s$, has data set $D_i$. The sizes of the data sets are *public*, as is
    the value $k$. Finally, the range of possible input values $[\alpha, \beta]$ 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**
3:     Set $m = \lceil (a + b)/2 \rceil$ and output it.
4:     Each party computes the number of elements in its data set that 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 $i$, respectively.
5:     The parties engage in the following secure computation:
       [**In the malicious case:** Verify for every party $i$ that $l(i) + g(i) \le |D_i|$, $l(i) \ge$
       $L(i)$, and $g(i) \ge G(i)$. In addition, if $m = \alpha$, then we check that $l(i) = 0$; if
       $m = \beta$, we verify that $g(i) = 0$.]
       In both the semi-honest and malicious cases:
       • If $\sum l(i) \le k - 1$ and $\sum g(i) \le n - k$ then "done". (This means that $m$ is
         the $k$th-ranked item.)
       • If $\sum l(i) \ge k$ then set $b = m - 1$. (This means that the $k$th-ranked element
         is smaller than $m$.)
         [**In the malicious case:** Set $G(i) = |D_i| - l(i)$. Note that as the right end-
         point of the range decreases, $G(i)$ is non-decreasing. This can be seen by
         noting that $|D_i| - l(i) \ge g(i)$, which is enforced to be at least as much as
         the previous value of $G(i)$. (Since the left end-point of the range remains
         the same, $L(i)$ remains unchanged.)]
       • If $\sum g(i) \ge n - k + 1$ then set $a = m + 1$. (This means that the $k$th-ranked
         element is larger than $m$.)
         [**In the malicious case:** Set $L(i) = |D_i| - g(i)$.]
6: **until** "done"
```

Correctness The correctness of this algorithm follows from observing that if m is the kth-ranked element then the first condition that is checked, namely whether $\sum l(i) \le k - 1$ and $\sum g(i) \le n - k$, is met and the output of the algorithm is defined to be m. Otherwise, the kth-ranked element is in the reduced range that the algorithm retains for its next iteration.

Overhead The number of rounds is $\log M$. Each round requires a secure multi-party computation that computes two summations and performs two comparisons. A circuit computing the sum of s $\log M$-bit numbers is of size $O(s \log M)$, while a comparison can be computed using $O(\log M)$ gates. Therefore, the size of the circuit used in each round is $O(s \log M)$, and this is also the number of its input bits. The secure evaluation can be implemented, for example, using generic protocols for secure multi-party computation, such as the protocols of [3,13,16]. In the malicious adversary case, the computation can be implemented using the constructions of [8,10] for a secure computation of a reactive system (alternatively, the system can use a protocol similar to the one we describe in Appendix 4.1 for the two-party case).

4.1. *Security*

Theorem 4. *Protocol* FIND-RANKED-ELEMENT-MULTIPARTY *securely computes the kth-ranked element of $D_1 \cup \cdots \cup D_s$, for the case of a semi-honest adversary.*

Proof. The proof is based on the composition theorem which shows that it is sufficient to consider the hybrid model where the multi-party computation in step 5 is done by a trusted party. We show that in this case the protocol is secure against an adversary that controls up to $s - 1$ of the parties. (If we implement the multi-party computation of Step 5 by a protocol which is secure against an adversary that controls up to t parties, e.g., using [3,13,16], it follows from the composition theorem that the resulting protocol is secure against an adversary which controls up to t parties.)

Consider an adversary operating in the hybrid model and controlling a subset of the players. The output it can assign to these players is a function of the output of the protocol (i.e., the kth ranked element), and of the results of the comparisons done in Step 5 of the protocol. We show that there is an equivalent adversary B operating in the ideal model, that only receives the kth ranked element, and can compute the results of the comparisons done in Step 5 of the protocol. Therefore, B can also compute the output computed by A.

Indeed, knowing the range $[a, b]$ that is used at the beginning of a round, B can compute the target value m used in that round. If m is the same as the output, B can conclude 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, and outputs the corresponding answer to the comparison done in Step 5 (if the real execution had not done the same, the kth ranked element 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 B can simulate the execution of the protocol and compute the answers of the comparisons done in Step 5. $\square$

Theorem 5. *Protocol* FIND-RANKED-ELEMENT-MULTIPARTY *securely computes the kth-ranked element of $D_1 \cup \cdots \cup D_s$, in the presence of a malicious adversary which controls all but one of the parties.*

Proof. The proof in this case, too, is based on the composition theorem, which shows that it is sufficient to consider the hybrid model where the multi-party computation in step 5 is done by a trusted party. The secure implementation of this computation can be done using the constructions of [8,10] for a secure computation of a reactive system.[5]

The proof shows that for every adversary that corrupts up to $s-1$ parties in the computation in the hybrid model, there is an adversary which corrupts these parties in the ideal model, and which results in a communication transcript which is indistinguishable from that of the former adversary. 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 outcome is legitimate in the ideal model). Moreover, we assume that the result of the computation of Step 5 is first given to Party 1, and therefore both model an adversary which controls this party can learn the output of the function and abort before the legitimate players learn it.

The proof shows how to examine the operation of the adversary in the hybrid model and generate input for the adversary in the ideal model, which it then gives to the trusted party. Given the output of the trusted party it is then easy to simulate the operation of the protocol, as described in the proof of Theorem 4.

Imagine a tree of size M with each node in the tree corresponding to a guess m (in the protocol) for the value of the median. The root corresponds to the initial guess $m = m_0 = \lceil(\beta - \alpha)/2\rceil$ with the initial range being $[\alpha, \beta]$. Its left child corresponds to the next guess for m if the first guess is incorrect and the median is discovered to be smaller than m_0 and is in the range $[a, b]$, with $a = \alpha$ and $b = m_0 - 1$. Similarly, the right child corresponds to the next guess for m if the median is discovered to be larger than m_0, and is in the range $[m_0 + 1, \beta]$. The entire tree is constructed recursively in this manner. The leaves are associated with ranges containing a single integer. Note that each integer in the interval $[\alpha, \beta]$ is associated with the single node in the tree at which the guess m is set to the value of this integer. We will overload m and use it to refer to this node as well.

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. (In the following analysis, we examine the values given by a single party, party i, controlled by the adversary, but the same analysis can be applied to any subset of parties.) Consider a node u associated with the guess u. Then, the two values $l_u(i)$ and $g_u(i)$ that party i provides to the comparison executed at node u are supposed to be the number of items in the input of party i which are smaller than and larger than u, respectively. Also, let us denote by $e_u(i) = |D_i| - l_u(i) - g_u(i)$ the number of items that are specified by the adversary to be equal to u. Note that the values $l_u(i), g_u(i)$ are reported by party i during the execution of the protocol, while the value $e_u(i)$ is implied by the former values. We first show how we can run and rewind the adversary to learn all the $e_u(i)$ values. We then show that the adversary's behavior is completely consistent with the interpretation that $e_u(i)$ is the number of items with value u. Then this set can be used by the adversary in the ideal model as an input to the trusted third party and the output used to produce a computationally indistinguishable transcript.

[5] The basic idea of these constructions is that the parties run a secure computation of each step using, e.g., the protocol of [3]. 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.

In order to learn about the adversary, we 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. Our first goal is to show that for every node $\alpha \le u < \beta$ the reported values by party i satisfy the condition $e_u(i) = l_{u+1}(i) - l_u(i)$. Therefore it is possible to extract the $e_u(i)$ values from the $l_u(i)$ values used in the protocol execution.

The boundary checks ensure that the following properties hold for any three nodes L, A, R that appear in this order in an inorder traversal of the tree (i.e., where L appears in the subtree rooted by the left descendant of A, and R appears in the subtree rooted by A's right descendant):

- When executing Step 5 with respect to node A, if it is decided to set $b = m - 1$, namely move to the left subtree, then it is verified that $G(i) = |D_i| - l_A(i)$. In the following steps of the protocol the value of $G(i)$ is never increased. Then, when node L is examined, the protocol verifies (in Step 5) that $g_L(i) \ge G(i)$. Therefore it holds that $g_L(i) \ge |D_i| - l_A(i)$, i.e., that $|D_i| - g_L(i) \le l_A(i)$. Equivalently this means that $l_L(i) + e_L(i) \le l_A(i)$, namely that the number of elements reported by i to be equal to the value L or smaller than it is not greater than the number of elements reported by i to be smaller than A.
- In a symmetric way, the protocol ensures that $g_R(i) + e_R(i) \le g_A(i)$. Since $|D_i| = l_A(i) + e_A(i) + g_A(i) = l_R(i) + e_R(i) + g_R(i)$, this inequality implies that $l_A(i) + e_A(i) \le l_R(i)$.
- The previous two observations, $l_L(i) + e_L(i) \le l_A(i)$, and $l_A(i) + e_A(i) \le l_R(i)$, enforce that for any two nodes u and v with $u < v$ it holds that $l_u(i) + e_u(i) \le l_v(i)$ (this holds since for every pair u, v we can represent these nodes as $u = L$, and $v = A$ or $v = R$). In particular, it holds for every $u \in [\alpha, \beta - 1]$ that $l_u(i) + e_u(i) \le l_{u+1}(i)$. Summing over these inequalities gives $\sum_{u=\alpha}^{\beta} e_u(i) \le l_\beta(i) + e_\beta(i) - l_\alpha(i)$.
- Since $l_\alpha(i) = 0$ and $g_\beta(i) = 0$ (enforced by our checks), it holds that $\sum_{u=\alpha}^{\beta} e_u(i) \le l_\beta(i) + e_\beta(i) + g_\beta(i) - l_\alpha(i) = D_i$.
- A symmetric analysis shows that for every $u \in [\alpha + 1, \beta]$ it holds that $g_u(i) + e_u(i) \ge g_{u-1}(i)$. Summing over these inequalities and applying the same analysis as before we get that $\sum_{u=\alpha}^{\beta} e_u(i) \ge D_i$.
- Combining the two inequalities above results in $\sum_{u=\alpha}^{\beta} e_u(i) = D_i$. Thus, all the inequalities must be satisfied with equality, implying $e_u(i) = l_{u+1}(i) - l_u(i)$ for all $\alpha \le u < \beta$.

As for the time it takes to rewind the operation of the adversary in order to learn all $e_u(i)$ values, this time is linear in $\beta - \alpha$, the size of the range. The time can be improved to $O(|D_i|)$, the size of the input of party i, by observing that we are only interested in nodes u for which $l_u(i) \ne l_{u+1}(i)$. Therefore, when examining the tree and observing, for instance, that the $l(i)$ value provided for the left child of a node is equal to the $l(i)$ value of the node (i.e., $l_{\lceil \alpha + (u-\alpha)/2 \rceil}(i) = l_u(i)$), we do not have to explore any nodes in the range $[\lceil \alpha + (u - \alpha)/2 \rceil, u]$.

To complete the proof we note that the result of this examination defines the input that the corrupt party i provides to the trusted party in the ideal model. More specifically, we set the input to contain $e_u(i)$ items of value u, for every $u \in [\alpha, \beta]$. The trusted party computes the kth value and returns it to us. This value defines the path that real execution of the protocol takes in the tree defined above. Therefore, it is possible to simulate the answers that party i receives in each step of the execution the protocol (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 thank Yehuda Lindell and Kobbi Nissim for stimulating discussions on this work.

Appendix: Implementing Reactive Computation in the Two-Party Case

We show here the main components of a simple implementation of protocol Find-Ranked-Element for the malicious adversary case. Alternatively, the protocol could use the generic constructions of reactive computation provided in [8,10].

The protocol is implemented by evaluating the following circuit, using Yao's construction of a secure two-party computation for the malicious case adversary [16,25,32,33].

- The circuit is composed of two stages.
  - The first stage has j layers (assuming that Step 4 of protocol Find-Ranked-Element is run j times). It has $2j$ inputs $m_{A,1}, m_{B,1}, \dots, m_{A,j}, m_{B,j}$ (each $\log M$ bits long) and j single bit outputs $out_1, \dots, out_j$. (This stage corresponds to Step 4 of the protocol).
  - The second stage has four additional $\log M$ bit inputs. It has one $\log M$ bit output. It is used for implementing Step 5 of the protocol.
- The circuit has internal variables $l_{A,i}, l_{B,i}, u_{A,i}, u_{B,i}$ for layer i, $1 \le i \le j$, corresponding to the variables that are used in the protocol. These variables are used to transfer the state from layer to layer. The variables with index $i = 1$ are initialized as in the protocol.
- Each layer in the first stage corresponds to an application of Step 4 of the protocol, i.e., it receives inputs $m_{A,i}, m_{B,i}$, and checks that both are within the corresponding bounds with parameter i. If the comparison is fine, it compares $m_{A,i}$ to $m_{B,i}$ and outputs the result as out_i. In addition, the internal variables of layer $i + 1$ are updated based on those of layer i and on the result of the comparison, as defined in the protocol. Figure 4 depicts a single such layer of the circuit. (Note that the output bit out_i depends only on inputs $m_{A,\ell}, m_{B,\ell}$ for $\ell \le i$. Therefore, the parties can provide this input to the execution of layer i.)

  Overhead: Each layer computes four comparisons in order to check that the inputs are in the required range, a single comparison between $m_{A,i}$ and $m_{B,i}$, and four additional comparisons in order to set the correct values for the bounds sent to the next layer. Each comparison is computed by a number of gates which is linear in the lengths of its inputs.
- The final layer implements the operation defined in Step 5 of the protocol, which is composed of a constant number of comparisons. Note that according to the protocol this layer must first apply the following test to A's input: if all previous comparisons yielded the result that $m_A < m_B$ (in which case $u_A = \infty$) then A's

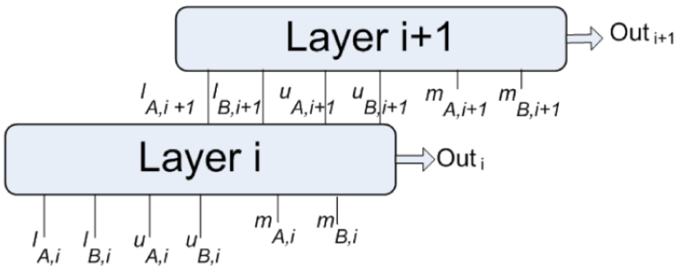


Fig. 4. A single stage of the circuit.

input must be greater than the last value of l_A. Otherwise, A's input must be equal to the last value of u_A. A similar test is applied to B's input.

The circuit is evaluated using Yao's protocol for secure two-party computation, secure against malicious parties [16,25,32,33]. Namely, one party (the circuit constructor) assigns garbled values to the wires, and the other party (the circuit evaluator) computes the circuit. The latter party uses oblivious transfer in order to learn the garbled values of its input wires.

The parties gradually provide the inputs to the layers. They first provide inputs $m_{A,1}, m_{B,1}$ to the first layer, observe its output and based on it provide the inputs $m_{A,2}, m_{B,2}$ to the second layer. Similarly, the parties must observe the output of the ith layer before providing the input to layer $i+1$.

Claim 1. *The circuit securely implements the functionality defined by Protocol* FIND-RANKED-ELEMENT.

Proof. The claim is proved by comparing the protocol to a protocol which is run in the hybrid model of [7], and observing that the two are equivalent. The latter protocol computes the function securely, based the composition theorem.

Let G_1 be a game in which the parties implement a secure two-party evaluation of the circuit described above. Let G_2 be a game in which each layer of the circuit is computed *separately*, where circuit C_i computes layer i, and the parties themselves communicate the "state" information from layer to layer while ensuring the confidentiality and authenticity of this information. This is done by C_i outputting shares of the state information, which are encrypted and authenticated by the secure computation of circuit C_i and are verified and decrypted by the secure computation of circuit C_{i+1}. In more detail, the state information that is generated by layer i includes the values of $l_{A,i+1}, l_{B,i+1}, u_{A,i+1}$ and $u_{B,i+1}$. It must be input to the computation of layer $i+1$, without revealing it to any of the parties, and while ensuring that none of the parties changes it. This is done using simple encryption and authentication as follows:

- Denote the state information output from layer i as $s_i = \langle l_{A,i+1}, l_{B,i+1}, u_{A,i+1}, u_{B,i+1}\rangle$, or s for short. The circuit chooses two random values s_A, s_B such that $s = s_A \oplus s_B$.[6] The output learned by A contains s_A, while B's output contains s_B.

[6] The circuit must therefore generate random output. This can be implemented by requiring each party to provide a random input to the circuit, and letting the circuit compute the exclusive-or of these inputs. If

- The circuit chooses two random values α_A, β_A and computes the value $r_B = \alpha_A \cdot s_B + \beta_A \bmod p$, where p is a large prime. It provides α_A, β_A as part of A's output, and r_B as part of B's output.

  Similarly, the circuit chooses two random values α_B, β_B and computes the value $r_A = \alpha_B \cdot s_A + \beta_B \bmod p$. It provides α_B, β_B as part of B's output, and r_A as part of A's output.
- The input for the circuit C_{i+1} computing layer $i+1$ contains $m_{A,i+1}, s_A, r_A, \alpha_A, \beta_A$ from party A, and $m_{B,i+1}, s_B, r_B, \alpha_B, \beta_B$ from party B. The circuit first verifies that $r_B = \alpha_A \cdot s_B + \beta_A \bmod p$ and that $r_A = \alpha_B \cdot s_A + \beta_B \bmod p$. If any of these checks fails then it aborts the computation. Otherwise it computes $s = s_A \oplus s_B$ and continues the computation as before.

It can be verified that in G_2, for every layer i, (1) none of the parties learns any information about the state s, and (2) if any of the parties tries to change the state that is transferred from layer i to layer $i+1$, the protocol aborts with probability $1 - 1/p$.

To show that G_1 and G_2 are indistinguishable, let us refine the distinction between the two games, and define game G_1^i as one in which the last i layers are implemented as in G_2, while the remaining layers as implemented as in G_1. G_1^0 is therefore equal to G_1, while G_i^{j+1} is equal to G_2. For any i, the computation of the two games G_1^{i+1} and G_1^i can only differ if one of the parties corrupts the state information passed between the two steps. This, however, happens with probability of at most $1/p$ for every state information. Therefore, assuming that $1/p$ is negligible, a hybrid argument shows that for every adversary attacking G_2 there is an adversary attacking G_1 which can generate an indistinguishable transcript with all but negligible probability.

We define G_3 as a game in which each of the secure computations of G_2 (i.e., computation of a circuit C_i) is replaced by a computation by a trusted third party, as in the hybrid model of [7]. By the composition lemma of [7]. This implementation surely securely implements the functionality defined by the protocol. The composition theorem states that if we replace the trusted party of game G_3 with secure protocols for computing each layer, then the composition of these protocols is secure. Namely, this means that the computation in G_2 is secure and consequently that G_1 is secure. □

References

[1] M. Atallah, M. Blanton, K. Frikken, J. Li, Efficient correlated action selection, in *Financial Cryptography* (2006), pp. 296–310

[2] D. Beaver, Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *J. Cryptol.* **4**(2), 75–122 (1991)

[3] D. Beaver, S. Micali, P. Rogaway, The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on the Theory of Computing* (1990), pp. 503–513

we do not want the parties to provide a random input which is as long as the randomness required by the circuit computation, the parties can be asked to input a random seed (say, a 128-bit long random string), and the circuit can then compute the exclusive-or of these seeds, and use the result as a seed to a pseudo-random number generator.

[4] I. Blake, V. Kolesnikov, Strong conditional oblivious transfer and computing on intervals, in *10th International Conference on the Theory and Application of Cryptology and Information Security ASIACRYPT* (2004), pp. 515–529

[5] C. Cachin, Efficient private bidding and auctions with an oblivious third party, in *Proc. 6th ACM Conference on Computer and Communications Security* (1999), pp. 120–127

[6] C. Cachin, S. Micali, M. Stadler, Computationally private information retrieval with polylogarithmic communication, in *Advances in Cryptology: EUROCRYPT '99* (1999), pp. 402–414

[7] R. Canetti, Security and composition of multiparty cryptographic protocols. *J. Cryptol.* **13**(1), 143–202 (2000)

[8] R. Canetti, Universally composable security: a new paradigm for cryptographic protocols, in *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science* (2001), pp. 136–145

[9] R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, R. Wright, Selective private function evaluation with applications to private statistics, in *Proceedings of Twentieth ACM Symposium on Principles of Distributed Computing* (2001), pp. 293–304

[10] R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai, Universally composable two party computation, in *34th ACM Symposium on the Theory of Computing* (2002), pp. 494–503

[11] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, R. Wright, Secure multiparty computation of approximations, in *Proceedings of 28th International Colloquium on Automata, Languages and Programming* (2001), pp. 927–938

[12] M. Fischlin, A cost-effective pay-per-multiplication comparison method for millionaires, in *RSA Security 2001 Cryptographer's Track*, vol. 2020 (2001), pp. 457–471

[13] M. Franklin, M. Yung, Communication complexity of secure computation, in *Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing* (1992), pp. 699–710

[14] P. Gibbons, Y. Matias, V. Poosala, Fast incremental maintenance of approximate histograms, in *Proc. 23rd Int. Conf. Very Large Data Bases* (1997), pp. 466–475

[15] O. Goldreich, *Foundations of Cryptography: vol. 2, Basic Applications* (Cambridge University Press, Cambridge, 2004)

[16] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game or A completeness theorem for protocols with honest majority, in *Proceedings of the 19th Annual Symposium on Theory of Computing*, May 1987, pp. 218–229

[17] S. Goldwasser, L. Levin, Fair computation of general functions in presence of immoral majority, in *Proceedings of Advances in Cryptology* (1991), pp. 77–93

[18] Y. Ishai, K. Nissim, J. Kilian, E. Petrank, Extending oblivious transfers efficiently, in *23rd Annual International Cryptology Conference* (2003), pp. 145–161

[19] H. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, T. Suel, Optimal histograms with quality guarantees, in *Proc. 24th Int. Conf. Very Large Data Bases* (1998), pp. 275–286

[20] S. Jarecki, V. Shmatikov, Efficient two-party secure computation on committed inputs, in *EUROCRYPT '07* (Springer, Berlin, 2007), pp. 97–114

[21] E. Kushilevitz, N. Nisan, *Communication Complexity* (Cambridge University Press, Cambridge, 1997)

[22] S. Laur, H. Lipmaa, Additive conditional disclosure of secrets and applications. Cryptology ePrint Archive, Report 2005/378, 2005

[23] H. Lin, W. Tzeng, An efficient solution to the millionaires' problem based on homomorphic encryption, in *Third International Conference Applied Cryptography and Network Security* (2005), pp. 456–466

[24] Y. Lindell, B. Pinkas, Privacy preserving data mining. *J. Cryptol.* **15**(3), 177–206 (2002)

[25] Y. Lindell, B. Pinkas, An efficient protocol for secure two-party computation in the presence of malicious adversaries, in *EUROCRYPT '07* (Springer, Berlin, 2007), pp. 52–78

[26] S. Micali, P. Rogaway, Secure computation, in *Proceedings of Advances in Cryptology* (1991), pp. 392–404

[27] M. Naor, K. Nissim, Communication preserving protocols for secure function evaluation, in *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing* (2001), pp. 590–599

[28] B. Pfitzmann, M. Waidner, Composition and integrity preservation of secure reactive systems, in *ACM Conference on Computer and Communications Security* (2000), pp. 245–254

[29] V. Poosala, V. Ganti, Y. Ioannidis, Approximate query answering using histograms. *IEEE Data Eng. Bull.* **22**(4), 5–14 (1999)

[30] M. Rodeh, Finding the median distributively. *J. Comput. Syst. Sci.* **24**(2), 162–166 (1982)

[31] L. von Ahn, N. Hopper, J. Langford, Covert two-party computation, in *Proceedings of the Thirty-Seventh Annual Acm Symposium on Theory of Computing* (2005), pp. 513–522
[32] A. Yao, Protocols for secure computations, in *Proceedings of the 23rd Symposium on Foundations of Computer Science* (1982), pp. 160–164
[33] A. Yao, How to generate and exchange secrets, in *Proceedings of the 27th Symposium on Foundations of Computer Science* (1986), pp. 162–167