# Filtering for Private Collaborative Benchmarking

Florian Kerschbaum[1] and Orestis Terzidis[1]

SAP Research, Karlsruhe, Germany
Florian.Kerschbaum@sap.com
Orestis.Terzidis@sap.com

**Abstract.** Collaborative Benchmarking is an important issue for modern enterprises, but the business performance quantities used as input are often highly confidential. Secure Multi-Party Computation can offer protocols that can compute benchmarks without leaking the input variables. Benchmarking is a process of comparing to the "best", so often it is necessary to only include the $k$-best enterprises for computing a benchmark to not distort the result with some outlying performances. We present a protocol that can be used as a filter, before running any collaborative benchmarking protocol that restricts the participants to the $k$ best values. Our protocol doesn't use the general circuit construction technique for SMC aiming to optimize performance. As building blocks we present the fastest implementation of Yao's millionaires' protocol and a protocol that achieves a fair shuffle in $O(\log n)$ rounds.

## 1 Introduction

Benchmarking is a management process where multiple companies evaluate their processes in comparison to each other, usually their competitors in their industry segment. Collaborative benchmarking is when multiple companies engage in this process together. Common statistical quantities, such as the average or variance, of business process performance quantities, e.g. time to ship, cash flow or return on investment, are used to compare performance. But many of the input variables to this stochastic calculation are very sensitive and highly confidential, even within one company. Gathering statistics over these variables is therefore a privacy-critical task. Current solutions solve this problem by anonymizing the data and use a trusted third party. Protocols that don't require a trusted third party are expected to increase customer acceptance.

Secure Multi-Party Computation (SMC) allows to compute such statistics without revealing anything about the input variables that cannot be inferred by the result. This paper focuses on enabling an important calculation for benchmarking. Often there is a small fraction of the participating companies whose performance is so outrageously bad that their inclusion in the benchmark, e.g. average, would distort the result and hamper the benchmarking process. Since benchmarking aims at process performance improvement, one could get a falsified picture of his standing compared to the competition.

We present a filter protocol that runs before the protocol that computes the statistical quantity and restricts the computation to the $k$ best values. Restricting to the $k$ best values is equivalent to excluding the $n - k$ worst values when $n$ is publicly known. It is general, because it can be applied before most other protocols that collaboratively compute benchmarks, e.g.[12], and that wants to exclude outlying values. The protocol is multi-party and each party holds one input value to the computation. The privacy requirements are that no one learns anything about anybody else's value, i.e. each value is kept private to its party. Also no one should learn anything about the $k$-partitioning of the values, i.e. no one should know whether anybody's - including his own - value is in the set of the $k$ included values or not.

The protocol sorts privately the input values: Each participant is assigned a rank $(1...n)$ and the idea is that at the end of the protocol, the values are sorted, such that the $i$th-ranked element is at the $i$th participant. The protocol emulates a sorting network [10] where each participant is connected to one input wire. The only operation in a sorting network is the comparison of two values at two participants, and eventually exchanging them. In a sorting network there are many comparison gates that are arranged in layers. Executing all comparison gates will sort the values. Each comparison gate performs the comparison between two parties' values and exchanges them if necessary. We are using an implementation of Yao's millionaire's protocol to protect the values during the comparison. Yao's millionaires' protocols make the result of the comparison public and in order to avoid leaking information in the sorting network we shuffle the input values with a random permutation unknown to all participants. Then the result of the comparison is a random variable and no one can track his own value through the sorting network. The only cryptographic tools our protocol uses are mix networks [9] and homomorphic encryption [21, 22].

The remainder of this paper is organized as follows: The next section reviews related work, section 3 presents one comparison step in the sorting network, section 4 shows how to start the protocol using mix networks and section 5 concludes the paper.

## 2 Related Work

There are few business-oriented applications of SMC related to our application of SMC to benchmarking in the literature. Specifically for benchmarking there is only one [7]. It presents a protocol to compute division with a secret divisor. It is extended to a number of useful protocols for benchmarking and forecasting and is an ideal candidate to be applied after our filter protocol. Another business application of SMC has been presented in [2]. Protocols for secure supply chain management are defined that protect a retailer such that its profit is not consumed by the supplier. Most protocols are simple, yet well motivated for this business application.

There are several protocols that solve algorithms privately that are related to our sorting problem. There is a protocol in [3] that finds the maximum of two additively split vectors. It uses homomorphic encryption and a protocol for Yao's millionaire's problem. In [1] the $k$th ranked element is computed. The solution for the two-party case is very clever, if $k$ is close to the median. It uses a solution to Yao's millionaire's problem. Its multiparty solution guesses the element by searching over its domain. Frikken et al. present solutions to private binary sort [12]. In several of their protocols they use solutions to Yao's millionaire's protocol.

Our protocol is also not the first to use mix networks for SMC. There is a class of protocols for private distributed constraint solving that use mix networks [25, 26]. For general SMC circuit constructions Jakobsson and Juels present a solution using homomorphic encryption and mix networks [19]. In addition to the examples above there are many SMC protocols not listed here that use homomorphic encryption.

Mix networks were invented in [9]. Many different cryptographic protocols have been derived from it. The idea of [9] has been put to practice for anonymous communication in [28]. The research in this area is very active and there are many more excellent results also not listed here.

Homomorphic encryption is available for many homomorphisms. We need a homomorphism over the addition group and two encryption schemes that can achieve that in practice are [21, 22].

We have avoided general SMC constructions, even though there are clever results on special protocols such as Yao's millionaire's problem. In [8] a protocol using a third party and a clever number theoretic construction is presented. Homomorphic encryption solves the problem in [11]. Although [11] is the best two-party solution known, it is still linear in the number of bits. Another cryptographic tool we have avoided for performance reasons in our protocol is Oblivious Transfer [23] which can be used to solve any SMC problem.

SMC was introduced in [30]. [30] also presents a general solution and Yao's millionaire's problem in which two millionaires want to compare their wealth, but do not want to reveal the exact number. Clever general constructions for SMC have been found in [6, 14]. Goldreich extends their presentation into an excellent expose [13]. The general idea is to construct a binary circuit of the function and evaluate it obviously. This can be done in one round, but the constructed circuit can be quite large. There exists a practical implementation of this general solution for two-party problems [20]. Nevertheless it is argued that for practical problems, faster solution are sought [15].

## 3 A Comparison Gate in the Sorting Network

Our protocol emulates a sorting network where each participant is connected to one input wire. The sorting then proceeds by executing the "compare and exchange" gates for all pairs of input wires. The gates in one layer of the network can be executed in parallel. Most practically efficient sorting networks have a depth of $O(\log^2 n)$ and a communication complexity of $O(n \log^2 n)$ and, since our comparison gates operate in $O(1)$ rounds, this step of the protocol can be completed in $O(\log^2 n)$ rounds with a communication complexity of $O(n \cdot \log^2 n)$. The comparison gates are preceded by a secret, random permutation protocol that can be of independent interest. Our protocol uses a solution to Yao's millionaires' problem that outperforms the best-known solution which can also be used in other contexts. The complexity of $O(\log^2 n)$ rounds and communication complexity of $O(n \log^2 n)$ of the overall protocol is therefore as efficient as the non-secure version.

## 3.1 Preliminaries

**Security Model** Our protocol works in the semi-honest or honest-but-curious model [13]. Each party follows the protocol as specified, but keeps a record of the messages and tries to gain as much information as possible from them.

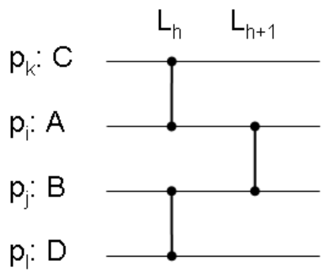

**Fig. 1.** A section of the sorting network

**Definitions** Let $p_1, \ldots, p_n$ be the participants of the protocol and $\boldsymbol{x} = (x_1, \ldots, x_n)$ be their input values, i.e. $p_i$ has input value $x_i$ for $i = 1, \ldots, n$.

Figure 1 shows a section of a sorting network. We compare the input values of the participants $p_i : A$ (Alice) and $p_j : B$ (Bob). Without loss of generality we will assume that $i < j$. We also assume that $k \neq j$ and $i \neq l$, since duplicate gates in subsequent layers are superfluous and can safely be removed. Furthermore, we assume, also without loss of generality, that $k < i$ and $j < l$, as depicted in figure 1.

We use public-key, semantically secure, homomorphic encryption that is homomorphic in the addition group (modulo some $m$). We denote such encryption with the public key of Alice as $E_A(\cdot)$ and decryption as $D_A(\cdot)$. The homomorphic property then states that there is an operation $\times$, such that $E_A(x) \times E_A(y) = E_A(x+y)$, and an operation $\star$, such that $E_A(x) \star y = E_A(x \cdot y)$. We used Paillier's encryption scheme [22] for implementation where $\times$ is (modular) multiplication and $\star$ is (modular) exponentiation. We assume that the public key of each participant in the homomorphic encryption scheme is known to all participants.

The comparison gate protocol interlocks the variables from several gates and we use this notation to differentiate variables from previous gates to newly introduced ones. We use subscript $[CA]$ to denote variables from the comparison gate between Charlie ($C$) and Alice ($A$), e.g. $r_{[CA]}$ is the variable $r$ from that gate. We do not write the subscript $[AB]$ for variables introduced for the current comparison between Alice and Bob. Furthermore each participant has one input value, i.e. Alice has $a$, Bob has $b$, Charlie has $c$ and so on. The comparison at each gate is between those input values, e.g. in the comparison gate between Charlie and Alice compares $c$ and $a$.

## 3.2 Protocol

Let, Alice ($A$) and Bob ($B$) be the two participants of the comparison gate protocol. Then Alice has value $a$, and Bob has value $b$. The goal of the comparison gate is to compute $a < b$ and eventually exchange them.

The privacy requirement is that neither Alice nor Bob may learn their value, since they flow through the sorting network. Therefore each input value $a, b$ is split between the two participants using addition, such that no party can infer a value by its local view of its share. This means, that Alice has the shares $a_A$ and $b_A$ and Bob has the shares $a_B$ and $b_B$ and that $a = a_A + a_B$ and $b = b_A + b_B$.

Alice and Bob can still compare the values.

$$a < b \Leftrightarrow a_A - b_A < b_B - a_B$$

The communication consequence of this splitting is that the predecessors need to transmit shares to the participants. Consider, the scenario in figure 1. Alice and Bob are engaging in the comparison gate protocol and have done so previously with Charlie ($C$) and Donna ($D$) (Alice with Charlie and Bob with Donna). From the previous comparison gate protocols Charlie and Donna, have shares of Alice's and Bob's values left. So, Charlie must send his share $a_{C[CA]}$ of Alice's input value to Bob and Donna her share $b_{D[BD]}$ of Bob's value to Alice. They become $a_B$ and $b_A$, respectively. We will present in the next section how Alice and Bob can do the comparison using a Yao's millionaires' protocol. After the comparison Alice and Bob need to eventually exchange the values depending on the result. They can do so by exchanging their local share, i.e. no interaction is necessary. The entire protocol is summarized in figure 2.

## 3.3 Yao's Millionaires' Protocol

The basic idea of our approach is to hide the difference by a hiding factor. To efficiently hide a number of size $O(m)$ by multiplication the random hiding factor has to be of size at least $O(m^2)$. We want to preserve the greater-than relation, so we have to prevent "wrap-around" modulo $n$. Negative values are

| Participants | Message / Operation |
|---|---|
| C $\longrightarrow$ B | $a_B = a_{C[CA]}$ |
| D $\longrightarrow$ A | $b_A = b_{D[BD]}$ |
| A $\longleftrightarrow$ B | $\rho = \mathsf{Yao}(a_A - b_A, b_B - a_B)$ |
| A | if $\neg\rho$ then swap($a_A$,$b_A$) |
| B | if $\neg\rho$ then swap($a_B$,$b_B$) |

**Fig. 2.** The protocol for one comparison gate

not represented in modular arithmetic, we therefore define the upper half of the range $[0, n-1]$ to be negative numbers:

$$[\lceil \frac{n}{2} \rceil, n-1] \equiv [-\lfloor \frac{n}{2} \rfloor, -1]$$

The multiplicative hiding has a draw-back, if the difference of $a$ and $b$ is 0, i.e. they are equal. Then the result of the hiding will be 0 regardless of the chosen hiding factor. This can be avoided by subtracting another (positive) random number that does not change the result, i.e. that is strictly smaller than the multiplicative hiding factor. The entire protocol is listed in figure 3.

1. Alice sends $E_A(a)$ to Bob.
2. Bob chooses random numbers $r$ and $r'$ with $0 \leq r' < r$.
3. Bob computes $E_A(c) = E_A(a)^r \cdot E_A(-r \cdot b + r') = E_A(r \cdot a - r \cdot b + r')$.
4. Bob sends $E_A(c)$ to Alice.
5. Alice decrypts $c = D_A(E_A(c))$ and decides $a < b$ if and only if $c \geq \frac{n}{2}$. The following derivation shows this equivalence:

$$c \bmod n \geq \frac{n}{2}$$

$$c < 0$$

$$r \cdot (a - b) + r' < 0$$

$$a - b \leq -1 < -\frac{r'}{r} < 0$$

6. Alice sends the bit $a < b$ to Bob.

**Fig. 3.** Yao's millionaires' protocol

If the numbers $a$ and $b$ to be compared are drawn from the domain $\mathcal{D}_a = [l_a, h_a]$, then the difference is in the domain $\mathcal{D}_- = [l_a - h_a, h_a - l_a]$. We can then choose the random numbers $r$ from the domain $\mathcal{D}_r = [l_r, h_r] = [1, (h_a - l_a)^2]$ and the random numbers $r'$ from the domain $\mathcal{D}_{r'} = [0, r]$. One can randomly

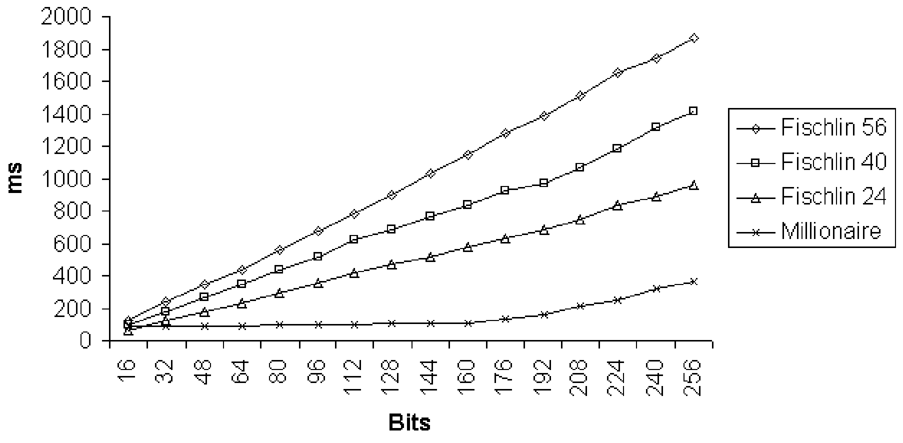

**Fig. 4.** Performance results

choose one of many possible distributions for choosing the numbers to increase the Alice's difficulty of guessing $b$. To run the protocol correctly the modulus $n$ of Paillier's encryption scheme needs to be larger than $2 \cdot ((h_a - l_a)^3 + (h_a - l_a)^2)$ to prevent "wrap-around".

There is a small leak in the protocol that occurs with very minor probability. If $c$ is lower than $l_r$, the lower bound of domain $\mathcal{D}_r$ (, i.e. $c = 0$), Alice knows that $a = b$. If $r$ and $r'$ are chosen uniformly from $\mathcal{D}_r$ and $\mathcal{D}_{r'}$, respectively, then probability $p$ of accidentally revealing $a = b$ (if $a$ is indeed equal to $b$) is:

$$p = \sum_{i=1}^{h_r} \frac{1}{h_r} \cdot \frac{1}{i} \approx \frac{\ln h_r}{h_r}$$

For large numbers $h_r$ this probability is negligible. E.g. when comparing 160-bit numbers, $p < 2^{-314}$.

**Performance** We have implemented the best-known Yao's millionaires protocol by Fischlin [11] to compare the performance to ours. The implementation was done in Java [17] and evaluated on a computer with a 1.6 GHz Pentium Mobile processor and 1 GB of RAM running Windows XP using version 1.4.2 of Sun's Java SDK [27].

The implementation of Fischlin's scheme is based on his optimized version which already provided a significant speed-up compared to an earlier version using more re-randomization steps. The algorithm for computing the Jacobi symbol for decrypting is from [4]. No sanity checks were performed, e.g., on the messages allowing the decryption algorithm to be supplied with a message with Jacobi symbol $J(m|n) = -1$. Instead the performance of each implementation was optimized, as long as the security of the protocol was not violated. This reduced the decryption of the Goldwasser-Micali (GM) encryptions [16] to one Jacobi computation modulo an RSA prime factor per cipher-text.

We used a 512-bit RSA modulus for the GM encryption. For our scheme we used the following formula to compute the key length of Paillier's encryption scheme from the bit length $bits_m$ of the numbers to be compared: $bits_n = \max(512, (\lfloor \frac{bits_m}{32} \rfloor + 2) \cdot 32)$ which satisfies our requirement for the modulus above. The choice of a minimum key-length of 512-bit seems to be acceptable only for low security requirements, since RSA keys with more than 640 bits have been successfully factored [5], but we expect similar advantages for larger key sizes. The term "low security requirement" refers to the time it takes to break the encryption and not the amount of information revealed in an information-theoretic sense.

We optimized the implementation of our scheme in two ways. First, we used the pre-computation of the dividend for decryption as suggestion in [22]. Second, we saved one more modular exponentiation by not randomizing the encryption of $b$. Instead, we just multiply with $g^b$ and rely on the randomization done by the multiplication with $r$. This leaves the 5 modular exponentiations: two for encryption of $a$, one for multiplying with $r$, one for subtracting $b$ and finally one for decrypting $c$. All modular exponentiations done modulo $n^2$.

Our performance results for the comparison of the two protocols are depicted in figure 4. We have run Fischlin's scheme with three different parameters for the error of probability: $2^{-24}$, $2^{-40}$ and $2^{-56}$. They are denoted as Fischlin 24, Fischlin 40 and Fischlin 56 in the figure. Our scheme is denoted as Millionaire.

## 4 Bootstrapping the Protocol

In the protocol above Alice and Bob learn the result of the comparison, i.e. they can track a value's partial flow through the sorting network. In the worst case Alice or Bob learns the rank of her or his value. Note, that if neither Alice nor Bob know where their input values come from, they cannot deduce anything about the input vector $\boldsymbol{x}$. We can achieve this with an initial permutation of all input values that is known to no participating party. Such permutations have been used in other SMC protocols [25, 26], but we need our own method to prepare the values for the comparison gate protocol. Also our randomized construction achieves such a permutation in $O(\log n)$ expected steps compared to their $O(n)$ protocol and thereby keeps the $O(\log^2 n)$ complexity of the overall protocol. The overall communication complexity of this protocol is $O(n)$.

### 4.1 Setup

We use a mix network [9] as a sub-protocol. The construction presented in this paper fits our purposes very well. It has the following properties:

1. Bob does not know that a message came from Alice.

2. Bob has an anonymous reply channel $c_{[BA]}$ where he can send Alice an answer.

We denote sending a message from Alice to Bob over the mix network as $A \rightarrow_{mix} B$ and sending the anonymous reply as $A \leftarrow_{mix} B$. Additionally to the mix network and the point-to-point secret channels we assume the existence of a synchronous, authenticated broadcast channel $c_{broadcast}$.

Without loss of generality we assume that in the first layer of the network the comparisons are between the participants $p_{2i-1}$ and $p_{2i}$ for $i = 1, \ldots, \frac{n}{2}$. We denote the sets of odd and even numbered participants as $P_{odd}$ and $P_{even}$, respectively.

### 4.2 Permutation Protocol

We will first show how to achieve a random permutation of the participants where part of the permutation is still known to the participants. Then we will modify the protocol slightly, such that it computes a derangement with the same properties. A derangement is a permutation with no fixed points, i.e. no participant remains at its position. Then we will use that protocol to achieve a permutation of the input variables, such that no participant knows anything about the permutation and no input variable is traceable, i.e. they are re-randomized.

First we will show how to compute a random permutation $\Pi$, such that each participant $p_i$ only knows $\Pi(p_i)$, i.e. he knows his position in the permuted vector. The protocol $\mathcal{P}_{permutation}$ proceeds as follows:

1. Each participant $p_i$ who has no incoming partner announces itself as $p_i$ over the broadcast channel $c_{broadcast}$. Let $s_{free}$ be the set of those participants. Initially this set contains all participants. If it is empty the protocol halts.
$$p_i \rightarrow c_{broadcast} : p_i \qquad \text{if } \nexists x \Pi(x) = i$$
2. Each participant $p_{in}$ who has no outgoing partner chooses randomly a participant $p_{out}$ from $s_{free}$. He sends to $p_{out}$ a message $m$ over the mix network.
$$p_{in} \rightarrow_{mix} p_{out} : m \qquad \text{if } \nexists \Pi(in)$$
3. Let $s_i$ be the set of incoming message over the mix network at participant $p_i$. Each participant $p_i$ then chooses randomly one participant $p_j$ from $s_i$ (if $s_i$ is not empty). $p_i$ sends over the anonymous reply channel to $p_j$ the message *accept* and to all other members $p_k$ of $s_i$ the message *reject* (if there are any).
$$p_j \leftarrow_{mix} p_i : accept \qquad p_j \in s_i$$
$$p_k \leftarrow_{mix} p_i : reject \qquad \forall p_k \in s_i \wedge k \neq j$$
4. On the receipt of the message *accept* participant $p_{in}$ adds $\Pi(in) = out$ to the permutation.
$$accept \Rightarrow \Pi(in) = out$$

5. The protocol continues at step 1.

This protocol can take $O(n)$ rounds in the worst case, if every participant chooses the same partner $p_{out}$ every time. But, since this is a randomized algorithm, its expected running time is much more interesting.

Assume that each participant chooses its outgoing partner $p_{out}$ in succession, i.e. first $p_1$, then $p_2$, and so on until $p_n$. If each participant chooses a different partner the protocol finishes in this round. Let $E_i$ be the expected number of participants that have an ingoing partner when it has been participant $p_i$'s turn. We can compute $E_n$ as follows:

$$E_i = E_{i-1} + \frac{n - E_{i-1}}{n} = 1 + \frac{n-1}{n} E_{i-1} = \sum_{j=0}^{i-1} \left( \frac{n-1}{n} \right)^j$$

$$E_n = \lim_{i \to \infty} E_i \approx 0.6321 \cdot n$$

This means a constant fraction of the participants are expected to find their partners in each round. From the resulting recurrence $T(n) = 1 + T(n - E_n)$ we can conclude that the expected number of rounds is $O(\log n)$ and the overall communication complexity $O(n)$.

### 4.3 Derangement Protocol

The protocol $\mathcal{P}_{derangement}$ modifies $\mathcal{P}_{permutation}$ slightly, such that it computes a derangement. Recall, that a derangement is a permutation where no element remains at its place. The only modification is to step 2 of the protocol:

2. Each participant $p_{in}$ who has no outgoing partner chooses randomly a participant $p_{out}$ from $s_{free}$ that is not himself. If he finds such a partner, he sends to $p_{out}$ a message $m$ over the mix network. If there is only him who has no ingoing partner, he announces $fail$ over the broadcast channel and the protocol will restart with an empty permutation $\Pi$.

$$p_{in} \rightarrow_{mix} p_{out} : m \qquad \text{if } \nexists \Pi(in) \wedge out \neq in$$

$$p_{in} \rightarrow c_{broadcast} : fail \qquad \text{if } s_{free} = \{in\}$$

Protocol $\mathcal{P}_{derangement}$ can theoretically never terminate, since it is not guaranteed that a derangement is found. The probability $p$ that a random permutation is a derangement is $p = \frac{1}{e}$ [29]. We are not choosing a random permutation in $\mathcal{P}_{derangement}$, but are already tilting the odds towards a derangement by not picking oneself as a partner in step 2. Nevertheless we are upper bounded by the random permutation and the expected number of restarts is $O(1)$ and we can find a derangement in $O(\log n)$ expected rounds.

### 4.4 Final Protocol

We will now put the pieces together and construct a protocol $\mathcal{P}_{final}$ that can be run as the initiation protocol for the sorting network. First the participants choose a random derangement. The values are sent along that derangement twice. In the first step the source of the value is already hidden and the second step hides the target from the source. This results in a permutation (and not a derangement any longer) that is unknown to all participants, since it is based on a derangement and no party is source and target of a single message.

We need to take some steps in order to prepare for the comparison gate protocol. The input values (e.g. $a$ and $b$) need to be split between Alice and Bob. Recall, that the initial comparisons are between $p_{2i-1}$ and $p_{2i}$ for $i = 1, \ldots, \frac{n}{2}$. We apply the random permutation only between those fixed pairs, i.e. each "odd" participant sends his value to his "even" partner. He does so in a split fashion and encrypts the shares with the public key from another random party in $P_{odd}$, such that no intermediate (even) party may learn the entire share.

Since in the resulting permutation a party may receive its own value, we must prevent it from learning that fact by viewing its local shares. This is done by re-randomizing the shares with two random variables $r_a$ and $r_b$ at the intermediate party, which makes the resulting shares independent again. This party also has to re-encrypt the value, since the choice of encryption key may reveal the value as well. He does so by forwarding the encrypted value to the key owner (an "odd" participant) who responds with the re-encrypted value. The key owner only learns the split values, neither source nor target, (as any participant in the comparison gate protocol could), and therefore cannot infer anything about an input value.

The entire protocol $\mathcal{P}_{final}$ is as follows:

1. Each "odd" participant $p_{2i-1}$ chooses randomly a public-key from another participant $p_j$ in $P_{odd}$. He sends:

$$p_{2i-1} \longrightarrow p_{2i} : j, r_{2i-1}, E_{p_j}(x_{2i-1} - r_{2i-1}) \qquad j \neq 2i - 1$$

2. Each "even" participant $p_{2i}$ engages in the $\mathcal{P}_{derangement}$ protocol. They obtain the derangement $\Pi$ of the even participants.
3. Each "even" participant $p_{2i}$ sends over the mix network to his partner $p_{intermediate} = p_{\Pi(2i)}$:

$$p_{2i} \rightarrow_{mix} p_{intermediate} : j, r_{2i-1}, E_{p_j}(x_{2i-1} - r_{2i-1}), r_{2i}, E_{p_j}(x_{2i} - r_{2i})$$

$$intermediate = \Pi(2i)$$

4. Each intermediate participant $p_{intermediate}$ (which is every "even" participant) sends to $p_j$:

$$p_{intermediate} \longrightarrow p_j : E_{p_j}(x_{2i-1} - r_{2i-1}), E_{p_j}(x_{2i} - r_{2i})$$

5. Each contacted $p_j$ chooses randomly a public-key from participant $p_k$ in $P_{odd}$ (including himself) for each message. He returns to $p_{intermediate}$:

$$p_j \longrightarrow p_{intermediate} : k, E_{p_k}(x_{2i-1} - r_{2i-1}), E_{p_k}(x_{2i} - r_{2i})$$

6. Each intermediate participant $p_{intermediate}$ chooses the random values $r_a$, $r_b$. He computes:

$$E_{p_k}(a_A) = E_{p_k}(x_{2i-1} - r_{2i-1}) \times r_a = E_{p_k}(x_{2i-1} - r_{2i-1} + r_a)$$

$$a_B = r_{2i-1} - r_a$$

$$E_{p_k}(b_A) = E_{p_k}(x_{2i} - r_{2i}) \times r_b = E_{p_k}(x_{2i} - r_{2i} + r_b)$$

$$b_B = r_{2i} - r_b$$

7. Each intermediate participant $p_{intermediate}$ sends over the mix network to his partner $p_{last} = p_{\Pi(P_{intermediate})}$:

$$p_{intermediate} \rightarrow_{mix} p_{last} : k, a_B, E_{p_k}(a_A), b_B, E_{p_k}(b_A)$$

$$last = \Pi(intermediate)$$

8. Each last participant $p_{last}$ (which is again every "even" participant) sends to his partner in the sorting network $p_{last-1}$:

$$p_{last} \longrightarrow p_{last-1} : k, E_{p_k}(a_A), E_{p_k}(b_A)$$

9. Each participant $p_{last-1}$ chooses random values $r'_a$ and $r'_b$ to blind his shares and sends the result to participant $p_k$ (if necessary):

$$p_{last-1} \longrightarrow p_k : E_{p_k}(a_A) \times E_{p_k}(r'_a), E_{p_k}(b_A) \times E_{p_k}(r'_b)$$

10. Each contacted participant $p_k$ replies with the decrypted content to each message and $p_{last-1}$ de-blinds:

$$p_k \longrightarrow p_{last-1} : a_A + r'_a, b_A + r'_b$$

We have now achieved a random permutation between all (pairs of) participants and the sharing of input values is such that the processing of the sorting network can start. The content of each message is untraceable due to the intermediate node switching the keys, such that the encryption key does not give a hint, about the origin and re-randomizing the contents itself. Nodes contacted for decrypting the contents cannot make any deductions based on the contents, since they are blinded to the contents by random variables, and even if they interact with the contacting party in the next layer, they cannot infer possible values in $\boldsymbol{x}$.

## 5 Conclusion

We have shown how to sort the input variables of $n$ nodes using a protocol based on a sorting network. First, the values are permuted, such that no participant knows where its input value came from. Then each comparison gate is processed, such that no information is leaked due to the interlocking mechanism used by the protocol, although no protocol for Yao's millionaire's problem is being used.

After processing the sorting network, we can run any benchmarking algorithm and easily restrict to the $k$ best values by excluding the shares of the $n-k$ highest or lowest ranked participants. Just the input variables of the nodes containing the $k$ maximum (or minimum depending of the definition of "best") need not be included in the computation. No participants gains additional information about the input vector or the sorting.

We can compute the average of the $k$ best input variables in the following way (see [24]): First, run the sort protocol defined above. Then, the first participant chooses a random variable $r$ and sends $sum = r + a_A + b_A$ to the second participant. Each participant then adds his shares $sum = sum + a_B + b_B$, if they are still in the range of the $k$ best values, and forwards it to the next participant. The last participant forwards the result $sum$ to the first participant who broadcasts $sum - r$ to all participants. The average is $\frac{sum}{k}$. To compute the variance each participant subtracts the average from his input value and squares it. They run the protocol outlined above on the result to obtain the variance.

## References

1. G. Aggarwal, N. Mishra, and B. Pinkas. Secure Computation of the kth-Ranked Element. *Proceedings of EUROCRYPT*, 2004.
2. M. Atallah, H. Elmongui, V. Deshpande, and L. Schwarz. Secure supply-chain protocols. *Proceedings of the 5th IEEE International Conference on Electronic Commerce*, 2003.
3. M. Atallah, F. Kerschbaum, and W. Du. Secure and Private Sequence Comparisons. *Proceedings of the 2nd annual Workshop on Privacy in the Electronic Society*, 2003.
4. E. Bach, and J. Shallit. Algorithmic Number Theory. *MIT Press*, 1996.
5. F. Bahr, M. Boehm, J. Franke, and T. Kleinjung. RSA200. Available at *http://www.crypto-world.com/announcements/rsa200.txt*, 2005.
6. M. Ben-Or, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Proceedings of the 20th annual ACM symposium on Theory of computing*, 1988.
7. M. Bykova, M. Atallah, J. Li, K. Frikken, and M. Topkara. Private Collaborative Forecasting and Benchmarking. *Proceedings of the 3rd annual Workshop on Privacy in the Electronic Society*, 2004.
8. C. Cachin. Efficient private bidding and auctions with an oblivious third party. *Proceedings of the 6th ACM Conference on Computer and Communications Security*, 1999.
9. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, Vol. 24(2), 1981.

10. T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms, 2nd Edition. *MIT Press*, 2001.
11. Marc Fischlin. A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires. *RSA Security Cryptographer's Track*, 2001.
12. K. Frikken, and M. Atallah. Privacy Preserving Electronic Surveillance. *Proceedings of the 2nd annual Workshop on Privacy in the Electronic Society*, 2003.
13. O. Goldreich. Secure Multi-party Computation. Available at *http://www.wisdom.weizmann.ac.il/˜oded/pp.html*, 2002.
14. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. *Proceedings of the 19th annual ACM conference on Theory of computing*, 1987.
15. S. Goldwasser. Multi party computations: past and present. *Proceedings of the 16th annual ACM symposium on Principles of distributed computing*, 1997.
16. S. Goldwasser, and S. Micali. Probabilistic Encryption. *Journal of Computer and Systems Science 28(2)*, 1984.
17. J. Gosling, B. Joy, G. Steele, and Gilad Bracha. Java Language Specification, 2nd Edition. *Addison-Wesley*, 2000.
18. J. Groth. A Verifiable Secret Shuffle of Homomorphic Encryptions. *Proceedings of Practice and Theory in Public Key Cryptography*, 2003
19. M. Jakobsson, and A. Juels. Mix and Match: Secure Function Evaluation via Ciphertexts. *Proceedings of ASIACRYPT*, 2000.
20. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - A Secure Two-party Computation System *Proceedings of the 13th USENIX Security Symposium*, 2004.
21. D. Naccache, and J. Stern. A New Public-Key Cryptosystem Based on Higher Residues . *Proceedings of the 5th ACM Conference on Computer and Communications Security*, 1998.
22. P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *Proceedings of EUROCRYPT*, 1999.
23. O. Rabin. How to exchange secrets by oblivious transfer. *Technical Memo TR–81, Aiken Computation Laboratory*, 1981.
24. B. Schneier. Applied Cryptography, 2nd Edition. *John Wiley & Sons*, 1996.
25. M. Silaghi. Solving a distributed CSP with cryptographic multi-party computations, without revealing constraints and without involving trusted servers. *Proceedings of the 4th International Workshop on Distributed Constraint Reasoning*, 2003.
26. M. Silaghi. Meeting scheduling system guaranteeing n/2-privacy and resistant to statistical analysis (applicable to any DisCSP). *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, 2004.
27. Sun Microsystems. J2SE 1.4.2 SDK. Available at *http://java.sun.com/j2se/1.4.2/*, 2005.
28. R. Dingledine, N. Mathewson, P. Syverson. Tor: The Second Generation Onion Router. *Proceedings of USENIX Security Symposium*, 2004.
29. N. Sloane. The On-Line Encyclopedia of Integer Sequences. Available at *http://www.research.att.com/˜njas/sequences/*, 2005.
30. A. Yao. Protocols for Secure Computations. *Proceedings of the annual IEEE Symposium on Foundations of Computer Science* 23, 1982.