Comparison-based MPC in Star Topology

Gowri R. Chandran[1], Carmit Hazay[2], Robin Hundt[1] and Thomas Schneider[1]
[1]*TU Darmstadt, Germany*
[2]*Bar-Ilan University, Israel*

Keywords: Secure Multi-party Computation, k^{th} Ranked Element, Scheduling Problems, Optimisation Problems.

Abstract: With the large amount of data generated nowadays, analysis of this data has become eminent. Since a vast amount of this data is private, it is also important that the analysis is done in a secure manner. Comparison-based functions are commonly used in data analysis. These functions use the comparison operation as the basis. Secure computation of such functions have been discussed for median by Aggarwal et al. (EUROCRYPT'04) and for convex hull by Shelat and Venkitasubramaniam (ASIACRYPT'15).
In this paper, we present a generic protocol for the secure computation of comparison-based functions. In order to scale to a large number of participants, we propose this protocol in a star topology with an aim to reduce the communication complexity. We also present a protocol for one specific comparison-based function, the k^{th} ranked element. The construction of one of our protocols leaks some intermediate values but does not reveal information about an individual party's inputs. We demonstrate that our protocol offers better performance than the protocol for k^{th} ranked element by Tueno et. al. (FC'20) by providing an implementation.

1 INTRODUCTION

Data is being constantly generated by organisations as well as individuals. To leverage this massive volume of unstructured data, organisations seek to use data analysis techniques. Data analysis can benefit organisations in various ways. For example, businesses can learn more about their target group by running analysis on the consumers trends, multiple companies can run analysis on their combined data to compare various data points such as salaries of employees, the key performance indicator, etc., hospitals and healthcare companies analyse their data using artificial intelligence and machine learning to obtain faster and more accurate diagnosis. In addition to using the data for analytics, these organisations, at times, wish to keep the data private as it contains sensitive information. In all the above mentioned examples the data being analysed is sensitive. In cases such as that of healthcare companies, the sharing of patient's data is forbidden by law. The analysis therefore has to be done in a manner which does not reveal the inputs. This is where secure Multiparty Computation (MPC) comes into play.

For the past couple of decades, MPC has been a prominent field of research. Starting with the seminal works of (Yao, 1982; Goldreich et al., 1987; Beaver et al., 1990) it is still a widely researched topic with recent works like (Lindell et al., 2015; Wang et al., 2017; Choudhuri et al., 2020). The problem of secure MPC focuses on a group of parties that do not trust each other, but still wish to compute a function f of their inputs while keeping their inputs private. Namely, it allows a set of mutually distrusting parties to securely compute a function on their joint inputs without revealing anything about their inputs except what can be inferred by the output. In the real world, there are adversaries present that may act maliciously to gain more information than they are supposed to. Semi-honest adversaries follow the protocol as it is but try to learn more information from the messages. In cases where companies or organisations run the protocol, semi-honest security is a realistic assumption, as the organisations would not deviate from the protocol for their reputation's sake. Another factor that is considered for the construction of an MPC protocol is the number of parties that are corrupted by the adversary. In our work, we consider semi-honest adversaries and a dishonest majority, i.e., $n-1$ parties can be corrupted by the adversary.

One of the standard approaches to implement constant round MPC used in (Beaver et al., 1990; Ben-David et al., 2008; Kolesnikov et al., 2009; Lindell et al., 2015; Lindell et al., 2016) is by using Garbled Circuits proposed by Yao (Yao, 1982), by converting the function to be computed to a boolean circuit and privately evaluating the gates. Alternatively, the MPC protocol by Goldreich, Micali and Wigderson (Goldreich et al., 1987), which also uses a boolean rep-

Chandran, G., Hazay, C., Hundt, R. and Schneider, T.
Comparison-based MPC in Star Topology.
DOI: 10.5220/0011144100003283
In *Proceedings of the 19th International Conference on Security and Cryptography (SECRYPT 2022)*, pages 69-82
ISBN: 978-989-758-590-6; ISSN: 2184-7711

resentation, works by secret sharing the wire values amongst the parties. This protocol has been improved and implemented in (Choi et al., 2012; Boyle et al., 2021). All the aforementioned protocols are generic MPC protocols which can be used to implement any function f by converting it to a boolean circuit.

Another line of works considers the development of protocols for specific functions to achieve improved efficiency by exploiting the properties of the underlying function and optimising the concrete protocols accordingly. Several works have proposed protocols for specific functions such as private set-intersection (Hazay and Venkitasubramaniam, 2017; Inbar et al., 2018; Pinkas et al., 2018; Rosulek and Trieu, 2021) for finding the intersection of multiple sets, secure pattern matching (Hazay and Lindell, 2008; Hazay and Toft, 2010; Yasuda et al., 2013; Faust et al., 2013) for finding matching patterns in texts and RSA key generation (Frederiksen et al., 2018; Hazay et al., 2019; Chen et al., 2021) for generation of the RSA modulus. Relevant to our work, in (Garay et al., 2007; Damgård et al., 2007; Damgård et al., 2008; Kolesnikov et al., 2009; Couteau, 2016), protocols for the secure comparison of integers have been proposed using various techniques.

The computation of these specific functions can be optimised by reducing the function f to the computation of smaller/easier computable primitives. One method is to reduce the secure computation of f into the secure evaluation of a boolean circuit. Another technique is to reduce the function f to multiple instances of a smaller function and perform a secure computation of this primitive. The latter reduction technique is used in (Shelat and Venkitasubramaniam, 2015) to reduce f to the comparison function, which takes two integer inputs and returns 1 if the first input is smaller than the second and 0 otherwise. Here the authors present a two-party protocol for the computation of a class of functions, where the parties only interact for implementing the comparison function. This reduction results in a much more efficient protocol, as the parties only communicate to evaluate the comparison function.

The comparison-based functions considered in (Shelat and Venkitasubramaniam, 2015) are widely used for various data analytic purposes. They include functions such as finding the convex hull, finding the median, job scheduling problems, matroid optimisations and many more optimisation problems. One of the functions that we discuss in detail in our work is the convex hull. The convex hull of a set of points is the smallest convex set which contains all the points in that set. We also discuss job scheduling, which is an optimisation algorithm where multiple parties have jobs that require the use of a common resource and the jobs are assigned to the resource at a specific time.

These functions have important real world applications. For instance, the secure computation of the convex hull of a set is useful in tracking a disease epidemic, where the extent of the spread of a disease can be monitored without revealing all the locations of the infected patients. A two-party variant of this problem is discussed in (Shelat and Venkitasubramaniam, 2015) where the authors use the *Gift Wrapping Algorithm* for computing the convex hull. Scheduling problems, such as job scheduling, have many applications in settings where the resources are limited and more than one user wishes to use them. Secure job scheduling can be used in applications where the details of the job (e.g. duration, amount of resource used etc.) are to be kept private, for instance in booking appointments at a doctor's, where the time taken is kept private.

Another functionality that we consider is finding the k^{th} ranked element of the union of multiple sorted sets. This function has applications in financial and medical analysis. (Aggarwal et al., 2004) present a secure protocol for the computation of the k^{th} ranked element, where the k^{th} ranked element is computed using the binary search algorithm. Following that, a constant round protocol for this function is presented in (Tueno et al., 2020), where the protocol is presented in a star topology with all the parties communicating only with a dedicated server.

In instances where multiple organisations wish to jointly analyse their data, often the communication occurs via WAN connections, making communication the bottleneck for running the protocol, as most organisations have high computational power. Therefore, protocols with low communication complexity are crucial. One method to achieve this is by constructing the protocols in a star network topology. The parties would then only need to communicate to one central party. The central party has its own input for the computation and also interacts with all the other parties in a series of secure two-party computations. This eliminates the need for broadcast channels, thus reducing the communication complexity.

1.1 Our Contribution and Outline

In this paper, we explore the concrete efficiency of comparison-based functions, i.e. a class of functions that can be reduced to a secure computation of comparison of integers. We present two different protocols for this class of functions. The first is a generic protocol (see §3) for secure computation of a class of functions called the Greedy Compatible functions (see §2.1) where we extend the two-party protocol from (Shelat and Venkitasubramaniam, 2015). We implement a reduction technique to reduce the computation of the function to a multi-party computation of the minimum of n integers and propose an efficient new method for computing the minimum. Next, we give concrete instantiations (see §3.2) of a few functions demonstrating the practical applications of our protocol. To the best of our knowledge, there have not been any previous works on specific multi-party protocols for the other two problems, i.e. convex hull and job scheduling. We show that our multi-party protocol can be used for the computation of these functions and that it has better performance compared to specific approaches and to generic MPC protocols that can be used for any of these functions.

Lastly, we present an alternative way of computing the k^{th} ranked element (see §4) by using reduction techniques similar to the generic protocol. Our protocol is the first that uses these reduction techniques to compute this function in a star topology. If the protocol is instantiated in a star topology, the computation of this function can be reduced to a secure computation of a summation function. Then communication only occurs during the computation of the summations, and the remaining computations are done locally by the parties and the comparisons are done locally by the central party. In particular, we reduce the computation of the k^{th} ranked element to a computation of secure summation which is instantiated using a threshold homomorphic encryption scheme and is implemented in a star topology where one of the participating parties plays the central party which interacts with the rest of the parties.

We note that our protocol for the k^{th} ranked element leaks the intermediate result of the summations to the central party. This leakage can reveal the distribution of the data in the union of sets. For some applications, revealing the distribution of data is a tolerable leakage. A potential approach for protecting this leakage can be by using differential privacy by revealing only the differentially private leakage. Combining MPC with differentially private tools has been used previously in (Groce et al., 2019) to achieve cheaper private set intersection by allowing differentially private leakage. In (He et al., 2017) the problem of private record linkage with differentially private leakage is studied. Sometimes, allowing some leakage can result in a more efficient protocol. There have been many works that discuss this trade-off on privacy for better performance. In (Cash et al., 2013; Pappas et al., 2014) some information related to the search queries is leaked in order to achieve more efficient database search functionalities. We leave the idea of using differentially private leakage for future work.

1.2 Related Work

Here, we mention several closely related works. The development of general-purpose MPC started with (Goldreich et al., 1987) and is still a major area of research with seminal works like (Franklin and Haber, 1996; Cramer et al., 2001; Ben-Efraim et al., 2016; Lindell et al., 2016; Ananth et al., 2019). These protocols can be used to instantiate the function for finding the minimum integer, which is one of the major underlying computations of our generic protocol. If we use the multi-party protocol in (Ananth et al., 2019) to instantiate the computation of minimum function, the resulting communication complexity is $\mathcal{O}(\kappa nd \log n \log d)$ with 2 rounds, where κ is the security parameter, n is the number of parties, and d is the input size. This protocol uses functional encryption combiners to achieve a constant round multi-party computation protocol, and although introducing good asymptotic results, it is not practical enough for an implementation.

Often specific purpose protocols are developed to replace the use of generic MPC and improve the performance. There has been abundant work done to develop protocols specifically for the secure comparison of two integers. In (Damgård et al., 2007; Damgård et al., 2008), homomorphic encryption is used to build a protocol for the two-party comparison of integers. Using either of these protocols for computing the minimum, a communication complexity of $\mathcal{O}(nd(d+\kappa))$ is achieved with $\mathcal{O}(\log n)$ rounds. In (Garay et al., 2007), a two-party protocol for the comparison of integers is presented using the encryption scheme of (Cramer et al., 2001). Using this protocol to implement the minimum function the online communication complexity is $\mathcal{O}(nd)$ with a round complexity of $\mathcal{O}(\log n \log d)$. In (Couteau, 2016), the authors use a block decomposition technique to compare the blocks of the integer and execute the comparison with Oblivious Transfer as a building block. By implementing the minimum function using (Couteau, 2016), an online communication complexity of $\mathcal{O}(nd)$ is achieved with a round complexity of $\mathcal{O}(\log n \log\log d)$. We compare the efficiency of our implementation of the minimum function with that of instantiating it with any of the above protocols in Tab. 1.

In (Tueno et al., 2020), a constant round protocol for the computation of the k^{th} ranked element is presented. The protocol is presented in a star network topology where the clients interact with a server. Unlike this setup, the central party in our protocol is one of the parties participating in the protocol and also provides input. Tueno et al. present several protocols in (Tueno et al., 2020), using different building blocks. The protocol using the garbled circuit approach has a

Table 1: Comparison of complexities for the computation of minimum function for n parties, using generic multi-party protocols ((Franklin and Haber, 1996; Ananth et al., 2019)) or two-party comparison protocols ((Garay et al., 2007; Damgård et al., 2007; Damgård et al., 2008; Couteau, 2016)) with our garbled circuit-based approach. Here, κ is the security parameter, d is the length of the input and n is the number of parties.

	(Franklin and Haber, 1996)	**(Garay et al., 2007)**	**(Damgård et al., 2007; Damgård et al., 2008)**	**(Couteau, 2016)**	**(Ananth et al., 2019)**	**This work**
Offline comm.	—	$\mathcal{O}(\kappa nd/\log\kappa)$	—	$\mathcal{O}(\kappa d/\log\kappa)$	—	$\mathcal{O}(\kappa d)$
Online comm.	$\mathcal{O}(n^2)$	$\mathcal{O}(nd)$	$\mathcal{O}(nd(d+\kappa))$	$\mathcal{O}(nd)$	$\mathcal{O}(\kappa nd\log n\log d)$	$\mathcal{O}(\kappa nd)$
Rounds	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n\log d)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n\log\log d)$	2	$\mathcal{O}(\log n)$
Assumption	DDH	OT	DGK	OT	LWE	OT

Table 2: Comparison of protocols for secure computation of the k^{th} ranked element. We compare our protocol with the one in (Aggarwal et al., 2004) and the three protocols presented in (Tueno et al., 2020) that are based on Yao's garbled circuit (YGC) and additively homomorphic encryption(AHE). n is the number of parties, κ is the security parameter, d is the bit-length of the input, t is the threshold of the additive homomorphic scheme, and S is the range of elements in the database.

	(Aggarwal et al., 2004)	**(Tueno et al., 2020)**			**This work**
		YGC	**AHE1**	**AHE2**	
Comm.	$\mathcal{O}(n^2\log S)$	$\mathcal{O}(\kappa n^2)$	$\mathcal{O}(\kappa n^2 dt)$	$\mathcal{O}(\kappa n^2 dt)$	$\mathcal{O}(n\log S)$
Rounds	$\log S$	4	4	4	$\log S$

total communication complexity of $\mathcal{O}(\kappa n^2)$, while the protocol using an additively homomorphic encryption scheme has a communication complexity of $\mathcal{O}(\kappa n^2 dt)$, where t is the threshold of the homomorphic encryption scheme. Our work achieves a communication complexity of $\mathcal{O}(\kappa n\log S)$, where S is the range of elements in the database, and a round complexity of $\mathcal{O}(\log S)$ rounds. We provide a comparison of our work and the previous works on the k^{th} ranked element in Tab. 2.

2 PRELIMINARIES

In this section we define some basic cryptographic primitives that are used in our protocols.

2.1 Greedy Compatible Functions

A function f is said to be *Greedy Compatible* (Shelat and Venkitasubramaniam, 2015), if f on the union of given sets can be defined using two functions $\mathcal{F}_{\text{MIN}}$ and $\mathcal{F}_{\text{UPT}}$, such that these functions have a few specific properties as specified in definition 1.

Definition 1 (Greedy Compatible Functions)**.** *The necessary and sufficient conditions for a function f to be* Greedy Compatible *are:*

1. *Unique solution: Given inputs X_i, $i=[1,n]$ there is a unique solution.*
2. *Unique order: The output $(c_1,...c_l)$ is released in a unique order, i.e.,*

$$f(X_1,...,X_n)=(c_1,...c_l)$$

where $c_1=\mathcal{F}_{\text{UPT}}(\perp,\bigcup X_i)$ and $c_{i+1}=\mathcal{F}_{\text{UPT}}((c_1,...,c_j),\bigcup X_i)$ for $i=[1,l-1]$.

3. *Local updatability: The function $\mathcal{F}_{\text{UPT}}$ on the union of all sets can be computed by computing the function $\mathcal{F}_{\text{UPT}}$ on each set individually and then computing $\mathcal{F}_{\text{MIN}}$ on its result. Namely,*

$$\mathcal{F}_{\text{UPT}}((c_1,...,c_j),\bigcup X_i)=$$
$$\mathcal{F}_{\text{MIN}}(\mathcal{F}_{\text{UPT}}((c_1,...,c_j),X_1),...,\mathcal{F}_{\text{UPT}}((c_1,...,c_j),X_n)).$$

2.2 Oblivious Transfer

1-out-of-2 Oblivious Transfer (OT) is a two-party protocol run between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$. The sender $\mathcal{S}$ inputs a pair of l-bit strings $s_0,s_1\in\{0,1\}^l$ and $\mathcal{R}$ inputs a choice bit $b\in\{0,1\}$. At the end of the protocol, $\mathcal{R}$ learns the chosen string s_b, but nothing about the unchosen string s_{1-b}, whereas $\mathcal{S}$ learns nothing about the choice bit b.

Oblivious Transfer Extension. OT protocols require costly public-key cryptography, but their performance can be improved using OT extension (Ishai et al., 2003; Asharov et al., 2013). OT extension allows extending a few public key-based base OTs using only symmetric cryptography and a constant number of rounds.

2.3 Garbled Circuits

An efficient way to evaluate a boolean circuit C in a constant number of rounds is Yao's garbled circuit (Yao, 1986; Lindell and Pinkas, 2004). In this approach, the circuit constructor $\mathcal{S}$ creates a garbled circuit $\widetilde{C}$ as follows: for each wire W_i of the circuit, $\mathcal{S}$ randomly chooses two garbled values $\widetilde{w}_i^0,\widetilde{w}_i^1$, where $\widetilde{w}_i^j$ represents the value j of W_i. Further, for each gate G_i, $\mathcal{S}$ creates a garbled table $\widetilde{T}_i$ with the following property: given a set of garbled values of G_i's inputs, $\widetilde{T}_i$ allows to recover the garbled value of the corresponding G_i's output, but nothing else. $\mathcal{S}$ sends these garbled tables, called garbled circuit $\widetilde{C}$ to the evaluator $\mathcal{C}$. Additionally, $\mathcal{C}$ obliviously (via OT) obtains the garbled inputs $\widetilde{w}_i$ corresponding to the inputs of both parties. Now $\mathcal{C}$ can evaluate the garbled circuit by evaluating $\widetilde{C}$ gate by gate, using the garbled tables $\widetilde{T}_i$. Finally, $\mathcal{C}$

translates the garbled output into the output values given for the respective parties.

2.4 Gift-Wrapping Algorithm

The Gift Wrapping algorithm (Jarvis, 1973) for finding the convex hull of a set works as follows: the first point in the convex hull is the leftmost point of the set. From this point a vertical line is considered. Then this line is rotated in a clockwise direction until it touches another point in the set. The first point that touches this line is the second point of the convex hull. Then a vertical line is considered from this point and again rotated in a clockwise direction. This process continues till the last point that falls on the rotation line is the first point of the convex hull.

3 COMPARISON-BASED FUNCTIONS

Here we propose an extension to the two-party protocol in (Shelat and Venkitasubramaniam, 2015), where a secure protocol to compute a class of functions called Greedy Compatible functions(cf. §2.1) is discussed. We extend their protocol to the multi-party setting and propose optimisations for the multi-party computation of the minimum function.

The Greedy Compatible functions can be defined in an iterative manner with all the computations done locally by each party except for computing the minimum. After each iteration, the output is slowly released so that the final output of the computation is the tuple of outputs from each iteration. Furthermore, the output of each iteration is given as the input for the next iteration. The only step where the parties interact with each other is for computing the minimum. We propose to reduce the communication between the parties by constructing the protocol in a star topology. Thus the parties only communicate with one central party which eliminates the need for broadcast channels.

As discussed above, the parties first compute a part of the function locally and then the minimum together. Consequently, the protocol π_{GP} (Fig. 3) for computing the function f involves instantiating two sub-functionalities: first, the local update function $\mathcal{F}_{\text{UPT}}$ (Fig. 1) and second, the minimum function $\mathcal{F}_{\text{MIN}}$ (Fig. 2). The functionality $\mathcal{F}_{\text{UPT}}$ updates the input of each party for the computation of the minimum functionality. The output of $\mathcal{F}_{\text{UPT}}$ is the input of $\mathcal{F}_{\text{MIN}}$ for the next iteration of the protocol. $\mathcal{F}_{\text{UPT}}$ is computed locally by each party, where its inputs are the party P_i's set of elements X_i and the output of $\mathcal{F}_{\text{MIN}}$. The output of $\mathcal{F}_{\text{UPT}}$ is the pair (x_i, δ_i). $\mathcal{F}_{\text{MIN}}$ takes (x_i, δ_i) as input and computes the minimum of all values δ_i and then returns the x_i corresponding to the smallest δ_i.

Now, combining these two functionalities we describe the protocol π_{GP} for securely computing f. In the initialisation step, the parties locally compute their first input pair by calling the functionality $\mathcal{F}_{\text{UPT}}$ on the set X_i. Then the iteration begins; for the first iteration, the parties send their input pair (x_i^1, δ_i^1) to the functionality $\mathcal{F}_{\text{MIN}}$, which computes $min\{\delta_1^1, \delta_2^1, ..., \delta_n^1\}$ and returns some $c_1 = x_t^1$ corresponding to the smallest δ_t^1. Then the parties update their inputs for the next iteration by calling $\mathcal{F}_{\text{UPT}}$ on c_1 and X_i. The protocol runs for $j = 1, ..., l$ iterations, with l depending on the computed function. As mentioned earlier, the protocol releases the output slowly, i.e., after each iteration, the parties receive c_j, which is a part of the final output $(c_1, c_2, ..., c_l)$.

Functionality $\mathcal{F}_{\text{UPT}}$

This functionality is computed locally by each party P_i.
Input: The set of elements X_i and the outputs $(c_1, ..., c_j)$ obtained by the previous iterations.

- In the initial step, given input $(\perp, X_i)$, $\mathcal{F}_{\text{UPT}}$ computes party P_i's input for $\mathcal{F}_{\text{MIN}}$ for the first iteration denoted by (x_i^1, δ_i^1).
- In the j^{th} iteration, given input $((c_1, ..., c_j), X_i)$, $\mathcal{F}_{\text{UPT}}$ computes party P_i's input for $\mathcal{F}_{\text{MIN}}$ for the $(j+1)^{th}$ iteration denoted by $(x_i^{j+1}, \delta_i^{j+1})$.

Output: $(x_i^{j+1}, \delta_i^{j+1})$ where $x_i^{j+1} \in X_i$ and δ_i^{j+1} is the associated index.

Figure 1: Local update function.

Functionality $\mathcal{F}_{\text{MIN}}$

Parties $P_1, \ldots, P_n$ participate in this computation.
Input: Each party P_i sends the pair (x_i, δ_i), where δ_i is an integer.

- Compute $\delta_t = \min\{\delta_1, ..., \delta_n\}$.
- Sets $c = x_t$, where x_t has the corresponding index δ_t.

Output: c.

Figure 2: Minimum function.

Security. We state and prove the security of this protocol next.

Theorem 1. *The class of Greedy Compatible functions (cf. definition 1) is securely computed by protocol π_{GP} (Fig. 3) in the presence of semi-honest adversaries for $n \geq 2$ in the $\mathcal{F}_{\text{MIN}}$-hybrid.*

Proof. We prove the security of the protocol in a hybrid model, where the function $\mathcal{F}_{\text{MIN}}$ is computed by

Protocol π_{GP}

Input: Each party P_i has a set of distinct elements X_i.
The Protocol:

1. Each party P_i locally computes $(x_i^1, \delta_i^1) \leftarrow \mathcal{F}_{\text{UPT}}(\perp, X_i)$.
2. For $j = [1, l]$:
   (a) The parties call functionality $\mathcal{F}_{\text{MIN}}$ for inputs (x_i^j, δ_i^j) to obtain output c_j.
   (b) Each party P_i locally applies functionality $\mathcal{F}_{\text{UPT}}$ on its input $((c_1, ..., c_j), X_i)$, computing the output $(x_i^{j+1}, \delta_i^{j+1})$, which will serve as its input in the next iteration.

Output: $c_1, ..., c_l$.

Figure 3: Semi-honest protocol for a *greedy compatible* function f.

a trusted third party. Consider $\mathcal{A}$ to be an adversary that corrupts a subset $\mathcal{I}$ of the parties. Let $(c_1, ...c_l)$ be the final output of the computation. We construct a simulator S that generates the view of P_i, $i \in \mathcal{I}$. S is given P_i's input X_i and the output $(c_1, ..., c_l)$, then S works as follows:

1. Given X_i, $i \in \mathcal{I}$ and $(c_1, ..., c_l)$, the simulator S invokes the corrupted parties on their corresponding inputs.
2. S plays the honest parties' role against the corrupted parties on arbitrary sets of inputs.
3. In the j^{th} iteration, given the inputs $\{(x_i^j, \delta_i^j)\}_{i \in \mathcal{I}}$ to $\mathcal{F}_{\text{MIN}}$, S simulates c_j as the output of $\mathcal{F}_{\text{MIN}}$.

In this case, the view of the corrupted party in the simulation is identical to that in the real execution of the protocol. From the unique ordering property of the solution, the two views are identical. Hence, the protocol π_{GP} securely computes any function f in the presence of semi-honest adversaries. ■

Complexity. In protocol π_{GP}, communication occurs only during the execution of $\mathcal{F}_{\text{MIN}}$. Let $\mathcal{O}(C)$ be the communication complexity of $\mathcal{F}_{\text{MIN}}$ and l be the number of rounds of the protocol. Since the parties execute $\mathcal{F}_{\text{MIN}}$ once per round, the total communication complexity is $\mathcal{O}(lC)$. We discuss the total complexity of the protocol in §3.1.

3.1 Realising $\mathcal{F}_{\text{MIN}}$

Recall that during the execution of π_{GP}, communication between the parties only occurs during the instantiation of $\mathcal{F}_{\text{MIN}}$. To reduce this communication, we propose an efficient way of computing the minimum function by splitting the multi-party computation of $\mathcal{F}_{\text{MIN}}$ into a series of two-party computations. We achieve this by performing the comparisons pairwise. Thus, we implement $\mathcal{F}_{\text{MIN}}$ in a star network topology where all the parties interact with one central party (say P_1) and not with any other party. This implies that the protocol π_{GP} can be implemented in a star topology as well. We define the pairwise computation of $\mathcal{F}_{\text{MIN}}$ in Fig. 4.

Protocol $\mathcal{F}^2_{\text{MIN}}$

Parties $P_1, \ldots, P_n$ participate in this computation.
Input: Each party P_i sends the pair (x_i, δ_i), where δ_i is an integer.

Let a and b be variables.

- $(a, b) = (x_1, \delta_1)$.
- For $i = [2, n]$,
$$(a, b) = (x_i, \delta_i), \quad if \;\; \delta_i < b$$
- $c = a$

Output: Each party receives $c = x_t$ such that $\delta_t = \min\{\delta_1, ..., \delta_n\}$ for $t \in [1, n]$.

Figure 4: Protocol for minimum function using pairwise comparisons.

Correctness. The correctness follows directly from a linear search algorithm.

Instantiation. We instantiate the pairwise comparisons using garbled circuits and construct the protocol in a star topology. Therefore, all communications between the parties happen via party P_1.

Our protocol is based on the idea of mobile agents (Cachin et al., 2000) that chains together multiple garbled circuit computations. We consider party P_1 to be the originator of the mobile agents protocol. Then the remaining parties are the hosts. The protocol works as follows: P_1 generates the message for $n-1$ parallel OTs. The next party, i.e., P_2 constructs a circuit using the output of P_1 and sends it to P_3 via P_1. P_3 then generates a circuit using P_2's output and sends it to P_4 via P_1. The parties continue similarly till the last party P_n sends the circuit to P_1 and P_1 evaluates the final circuit to obtain the final output. This construction can be modified to a binary tree based structure, where parties P_{2i}, for $i = 1, ..., \lfloor n/2 \rfloor$ can parallely run the second step and send the circuits to P_{2i+1}, for $i = 1, ..., \lfloor (n-1)/2 \rfloor$. In the third step, parties P_{2i+1} construct their circuits based on the received circuits and send it to party P_{4i+1}, for $i = 1, ..., \lfloor n/4 \rfloor$. We can also view this communication pattern as an evaluation of a hypercube, as presented in (Inbar et al., 2018), where each party represents a vertex of the cube.

Complexity. We first discuss the complexity for computing $\mathcal{F}_{\text{MIN}}$ and then compute the total complexity of the protocol π_{GP}.

The number of rounds required for the computation of the minimum is $\mathcal{O}(\log n)$. To increase the computation efficiency, we use OT extensions instead of plain OTs. As the precomputation for the OT extensions can be done in parallel, the communication complexity of the precomputation is $\mathcal{O}(\kappa d)$, where d is the size of the input. The total number of comparisons performed for each minimum function is $n-1$, hence the total communication complexity is $\mathcal{O}(\kappa n d)$.

We can instantiate the minimum functionality using either specific two-party protocols, or specific multi-party protocol or even using generic multi-party protocols. We now compare our results with those of using existing protocols. If we instantiate the pairwise comparisons using a two-party protocol for integer comparison (Couteau, 2016), it would require $\mathcal{O}(\log n \log\log d)$ rounds and would have a communication complexity of $\mathcal{O}(nd)$. Using a generic MPC protocol (Ananth et al., 2019), we can compute the minimum value in a constant number of rounds, with a communication complexity of $\mathcal{O}(\kappa n d \log n \log d)$. In Tab. 1, we give a detailed comparison of the protocols.

Thus, we see that our garbled circuit-based approach gives the most efficient instantiation of $\mathcal{F}^2_{\text{MIN}}$ in terms of communication complexity.

3.2 Concrete Instantiations of f

As discussed in §3, the protocol π_{GP} computes a class of functions f. The main challenge in implementing π_{GP} is to define the functionalities $\mathcal{F}_{\text{UPT}}$ and $\mathcal{F}_{\text{MIN}}$, such that correctness still holds. For each f, the definition of these functionalities changes according to f. Now we discuss some examples of f in detail and show how we can define $\mathcal{F}_{\text{UPT}}$ and $\mathcal{F}_{\text{MIN}}$ to realise the functions. Specifically, we consider the following functions: convex hull of a set and job scheduling. Two-party protocol for computation of convex hull have been given in (Shelat and Venkitasubramaniam, 2015). We give the multi-party protocol for this function and also present a new protocol for secure job scheduling. We also discuss the computation of median of the union of n sets in the full version of our paper.

3.2.1 Convex Hull

The convex hull of a set of points is the smallest convex set, which contains all the points in that set. Suppose there are n parties, each having a set of points. Then the convex hull of the union of these sets is the smallest convex set that contains all the points of all the sets. There are various algorithms that are used to find the convex hull of a set. Here we consider the Gift Wrapping algorithm (Jarvis, 1973) (see §2.4) which is the most efficient algorithm for this function.

Now, consider n parties and suppose each party P_i has a set X_i, then our objective is to compute the convex hull of the union of these sets. Each element in X_i is a point on a plane which is represented as $p_i = (x_i, y_i)$ where x_i and y_i are the X and Y coordinate of the point, respectively. Now we apply the Gift Wrapping Algorithm and define the functionality $\mathcal{F}_{\text{UPT}}$.

- For $j = 1$, $\mathcal{F}_{\text{UPT}}(\perp, X_i) = (p_i^1, \delta_i^1)$, where p_i^1 is the leftmost point in the set X_i (i.e., the point with the smallest X-coordinate) and δ_i^1 is the X-coordinate of p_i^1.
- For $j > 1$, if $p_i \in \{c_1, ..., c_{j-1}\}$, then P_i sends `terminate` to $\mathcal{F}_{\text{MIN}}$. Else, $\mathcal{F}_{\text{UPT}}((c_1, .., c_{j-1}), X_i) = (p_i^j, \delta_i^j)$, where p_i^j is the point that makes the smallest clockwise angle (larger than zero) with the vertical dropped from c_{j-1} and δ_i^j is the magnitude of the angle between the line joining c_{j-1} and p_i^j and the vertical from c_{j-1}. (The next point in the convex hull is the point that makes the least clockwise angle with the point p_{j-1}. Hence, each party sets its input for the next iteration by comparing the angles.)

The functionality $\mathcal{F}_{\text{MIN}}$ is defined exactly as in Fig. 2. Thus, the final output is $(c_1, ..., c_l)$, where each c_j is a point of the convex hull. The number of iterations of the protocol is equal to the number of points on the convex hull.

If there are three collinear points in $\bigcup X_i$, the rotation line will touch two points at the same time, which will give the same angle δ_i for two points p_i, which does not satisfy the properties of f in definition 1. Hence, we assume that no three points in the set are collinear.

3.2.2 Job Scheduling

Job scheduling is an optimisation algorithm where multiple parties have jobs that require the use of a common resource and these jobs are assigned to the resource at a particular time. Secure job scheduling can be used in applications where the details of the job (e.g., duration, amount of resource used, etc.) are to be kept private, for example in a car-sharing service where the clients would like to protect the information about the usage of the car.

Here we consider a job scheduling problem with one shared resource and multiple jobs. Consider n parties, each party P_i having a set of jobs $J_i = \{b_i^1, ..., b_i^t\}$. The goal is to find the order in which to assign these jobs to a common resource R. We consider the Shortest Job First (SJF) algorithm for the scheduling as this algorithm gives the best average waiting time for the parties. We instantiate the generic protocol in Fig. 3 to realise the function. The functionality $\mathcal{F}_{\text{MIN}}$ runs exactly like in Fig. 2 and we define the functionality $\mathcal{F}_{\text{UPT}}$ as follows:

- For $j=1$, $\mathcal{F}_{\text{UPT}}(\perp, X_i)=(b_i,\delta_i)$, where b_i is the shortest job in J_i and δ_i is the completion time for b_i.
- For $j>1$, if $J_i \subseteq \{c_1,\ldots c_{j-1}\}$, then $\mathcal{F}_{\text{UPT}}((c_1,\ldots c_{j-1}),J_i)=(\perp,\delta_i)$ where $\delta_i=\infty$. If $b_i^t \in \{c_1,\ldots c_{j-1}\}$, then $\mathcal{F}_{\text{UPT}}((c_1,\ldots c_{j-1}),J_i)=(b_i,\delta_i)$ where $b_i \in J_i \setminus \{b_i^t\}$ is the smallest job with completion time δ_i. Else if $jb_i^t \in \{c_1,\ldots c_{j-1}\}$, then $\mathcal{F}_{\text{UPT}}((c_1,\ldots c_{j-1}),J_i)=(b_i,\delta_i)$ where $b_i \in J_i$ is the shortest job and its completion time is δ_i.

The final output is $(c_1,\ldots,c_j)$, which gives the order in which to assign the jobs to the resource R. The protocol runs for $N=\sum_i |J_i|$ iterations.

4 LEAKY k^{th} RANKED ELEMENT

Now we focus on one specific comparison-based function, namely finding the k^{th} ranked element of a set.

Recalling that protocol π_{GP} (in §3) computes comparison-based functions that possess the properties specified in definition 1, it can therefore be implemented for computing the k^{th} ranked element of a union of sets. In this section, we construct a special protocol for the computation of the k^{th} ranked element. The protocol discussed here has a smaller communication complexity than a generic protocol computing this function, as well as previous protocols for the computation of this function. We also see that using this specific protocol is more efficient than π_{GP} (from §3) for computing the k^{th} ranked element.

The functionality for finding the k^{th} ranked element for N parties is defined by $\mathcal{F}_k(D_1,\ldots D_N)\mapsto(x_k,\ldots,x_k)$, where x_k is the k^{th} element of the union over all input sets D_i. We present a Leaky k^{th} Ranked Element protocol π_k^{Leaky} (Fig. 7) that securely realises functionality $\mathcal{F}_k^{\text{Leaky}}$ (Fig. 6). We use the binary search algorithm for finding the k^{th} element. This protocol works iteratively and requires two summations and two comparisons per iteration. We construct the protocol in a star network topology, hence splitting the computations into a series of secure two-party computations. By doing this, we reduce the communication complexity, but leak the result of the summations to the parties. This leakage is discussed in detail in §4.1.

Now we present the Leaky protocol in detail. The functionality $\mathcal{F}_k^{\text{Leaky}}$ (Fig. 6) computes the k^{th} ranked element of the union of N sets. It takes inputs from the parties for computing the sum. The result of this addition is then returned to all the parties. Next, the functionality receives an input from party P_1. If this input is equal to the k^{th} ranked element, then $\mathcal{F}_k^{\text{Leaky}}$ sends x_k (the k^{th} ranked element) to all the parties. Otherwise, if the input is smaller than x_k, $\mathcal{F}_k^{\text{Leaky}}$ sends 1 to all the parties and if the input is greater, it sends 0. The functionality is called leaky as it reveals the intermediate sum in each iteration to all the parties. This provides additional information to the parties which otherwise an adversary could not compute from the output alone.

The protocol π_k^{Leaky}, which realises functionality $\mathcal{F}_k^{\text{Leaky}}$, works in iterations and as follows: in the precomputation phase, each party computes $m=[a+b/2]$ and counts the number of elements in its database that are smaller than m (and larger than m). The parties encrypt inputs using a threshold homomorphic encryption scheme and send them to P_1. Party P_1 computes the sum of these encrypted inputs and all the parties jointly decrypt the result and obtain the plaintext. Next, P_1 compares the result of the sum to conclude whether the k^{th} ranked element is smaller than m (or larger than m). Based on the result of the comparison, the value of m is updated for the next iteration. Note that each party only communicates with party P_1 throughout the execution. This reduces the communication cost as compared to the multi-party protocol given in (Aggar-

Functionality $\mathcal{F}_k$

Functionality $\mathcal{F}_k$ communicates with all parties $P_1,\ldots,P_n$ and an adversary Sim.

- Upon receiving D_i from each party, compute x_k, where x_k is the k^{th} ranked element in $\bigcup_{i=1}^{n} D_i$. Send x_k to the adversary Sim. If Sim responds OK, transmit x_k to all parties.

Figure 5: Functionality for computing the k^{th} ranked element of a union of n sets.

Functionality $\mathcal{F}_k^{\text{Leaky}}$

Functionality $\mathcal{F}_k^{\text{Leaky}}$ communicates with all parties $P_1,\ldots,P_n$ and an adversary Sim.

- Receives (x_i,y_i) from each party.
- Upon receiving m from party P_1, if $m=x_k$, where x_k is the k^{th} ranked element, the functionality sends x_k to all parties. If $m>x_k$, then $\mathcal{F}_k^{\text{Leaky}}$ sends 1 to all the parties. Else, if $m<x_k$, $\mathcal{F}_k^{\text{Leaky}}$ sends 0 to all parties.

Figure 6: Leaky functionality for computing the k^{th} ranked element of a union of n sets.

wal et al., 2004) that computes the same functionality.

Leaky k^{th} Ranked Element (π_k^{Leaky})

Input: Each party P_i has a database D_i. The rank k, the range of elements in the union of databases ($[a,b]$) and the size of each database ($|D_i|$) are public.
Primitives: A homomorphic encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ having a key generation protocol π_{Gen}.
Initial Phase: Each party P_i ranks its elements in ascending order. $N = \Sigma_i |D_i|$ is the total number of elements in $\bigcup D_i$.
Key Generation Phase: The parties engage in a semi-honestly secure protocol π_{Gen} to generate a public key pk and their respective shares of secret key sk_i of sk.
Local Computation Phase: Each party P_i does the following:

1. Computes $m = \lfloor (a+b)/2 \rfloor$.
2. Computes the number of elements (l_i) less than m and the number of elements (g_i) greater than m.

Multi-party Phase:

4. Each party P_i encrypts its masked inputs, $c_i = \mathsf{Enc}_{\mathsf{pk}}(l_i)$ and $c'_i = \mathsf{Enc}_{\mathsf{pk}}(g_i)$, and sends the ciphertexts to P_1.
5. P_1 computes $[L] = \sum_{i=1}^{n} c_i$ and $[G] = \sum_{i=1}^{n} c'_i$.
6. The parties jointly decrypt $[L]$ and $[G]$ to obtain the sums L and G, respectively. Party P_1 receives the decrypted values L and G.
7. P_1 does the following comparisons
   (a) If $L < k$ and $G \leq N - k$, then m is the k^{th} ranked element and P_1 sends Foundk to all the parties.
   (b) If $L \geq k$, then P_1 sends 1 to all parties. Then each party sets $b = m - 1$ and repeats the protocol from the local computation phase.
   (c) If $G > N - k$, then P_1 sends 0 to all parties and each party sets $a = m + 1$ and repeats the protocol from the local computation phase.

Output: x_k (the k^{th} element of $\bigcup D_i$).

Figure 7: Leaky protocol for passively secure computation of the k^{th} ranked element.

Correctness. We prove the correctness of the protocol in the following argument. Let $[a,b]$ be the range of elements in the union of all the sets and N be the number of elements in the union. Let x_k be the element at the k^{th} position in the union of the sets where the elements are arranged in ascending order. Let m_i be the value of m in the i^{th} iteration. In the i^{th} iteration, each party counts the number of elements smaller than m_i and the number of elements greater than m_i and the sum of these values from all the parties is computed respectively. Let L_i and G_i be the total number of elements smaller than and larger than m_i respectively, in the i^{th} iteration. Then three cases arise.

- If $x_k < m_i$, then $x_k \in [a, m_i - 1]$. Then the number of elements smaller than m_i is greater than the number of elements smaller than x_k, i.e., $L_i > k - 1$. Thus, m_{i+1} will be computed as $\left\lfloor \frac{a + m_i - 1}{2} \right\rfloor$ and the procedure is repeated.
- If $x_k > m_i$, then $x_k \in [m_i + 1, b]$. Then, the number of elements larger than m_i will be greater than the number of elements larger than x_k, i.e., $G_i > N - k$. Thus, m_{i+1} is computed as $\left\lfloor \frac{m_i + 1 + b}{2} \right\rfloor$ and the procedure is repeated with m_{i+1}.
- Now if $L_i < k$ and $G_i \leq N - k$, then $x_k \notin [a, m_i - 1]$ and $x_k \notin [m_i + 1, b]$, which implies $x_k = m_i$.

Hence, the protocol correctly computes the k^{th} element.

Security. The protocol π_k^{Leaky} securely realises $\mathcal{F}_k^{\text{Leaky}}$ in the presence of semi-honest adversaries for $n \geq 2$. We discuss the proof of security in detail in §B.

Instantiations of the threshold PKE. The threshold homomorphic encryption in protocol π_k^{Leaky} can be instantiated using any homomorphic encryption schemes (see §A). In the implementation of our protocol, described in §4.3, we use the threshold Paillier PKE (Paillier, 1999). The threshold Paillier can be implemented with distributed RSA modulus generation, as discussed in (Hazay et al., 2019).

4.1 Leakage

Now we discuss the leakage mentioned above in protocol π_k^{Leaky}. In each iteration of π_k^{Leaky}, the sum of the number of elements smaller than m (L) and the sum of the number of elements greater than m (G) is leaked to party P_1. Therefore for each m, the number of elements greater or smaller than m in the union of all databases, i.e., $\bigcup D_i$, is revealed. Using this leakage from each iteration, an adversary can calculate the number of elements that lie between two values of m. This shows the distribution of the elements in $\bigcup D_i$ and $\bigcup D_j$, for $j = [2, n]$. However, this leak only reveals a collective information about the union of the databases, and the distribution of elements in each individual set D_j cannot be computed from this leakage.

In some applications certain leakage may be tolerable as a tradeoff between privacy and efficiency. This can be demonstrated by several works like (Cash et al., 2013; Pappas et al., 2014; Kolesnikov et al., 2015; Schoppmann et al., 2018) that leak some information in order to obtain more efficient protocols. For instance, in (Cash et al., 2013; Pappas et al., 2014; Schoppmann et al., 2018) DBMS search protocols are presented that allow leakage of some information to improve the efficiency of the search. (Kolesnikov et al., 2015) studies the dual-execution paradigm (Mohassel and Franklin, 2006) where the efficiency of two-party computation is improved by revealing a single bit of the honest party's input to the adversary. These works demonstrate tradeoffs between privacy and efficiency, where some leakage may be accepted in order to achieve higher efficiency. Moreover, if the leakage is to be reduced, a potential solution may be to use differential privacy. Then the leakage in the protocols would be the differentially private leakage as demonstrated in (Groce et al., 2019).

4.2 Complexity

Let $S = b - a + 1$, where $[a, b]$ is the range of elements in $\bigcup D_i$. Then, the maximum number of rounds is $\log S$. The communication occurs at the setup phase for generating correlated randomness and the multi-party phase for finding the k^{th} element. The communication complexity of the key generation phase is $\mathcal{O}(\kappa \cdot n^2)$ where n is the number of parties participating. In the multi-party phase, in each round, the communication occurs for: n encryptions, 1 decryption and 1 broadcast by P_1. Hence, the communication complexity of the online phase protocol is $\mathcal{O}(\kappa n d \log S)$, where d is the length of the inputs.

The protocol in (Aggarwal et al., 2004) also uses the binary search algorithm and requires $\log S$ rounds. The circuit consists of two summations and two integer comparisons, therefore the circuit size is $\mathcal{O}(n \log S)$. Hence, using an efficient MPC protocol (Ananth et al., 2019) for the computation of the circuit, the total complexity for the protocol becomes $\mathcal{O}(\kappa n d \log n \log d \log S)$.

The protocol in (Tueno et al., 2020) uses a star topology to achieve a constant round protocol. Using the additively homomorphic encryption as the basis, they obtain a 4 round protocol. They compute the rank of the element by comparing each element with every other element. The communication complexity of their protocol is therefore quadratic in the number of participating parties, i.e., $\mathcal{O}(\kappa n^2 d)$. We give the comparison of the complexities of the above protocols with our work in Tab. 2.

4.3 Implementation

We implemented the protocol π_k^{Leaky} in the Rust programming language and instantiated the threshold homomorphic encryption with the threshold Paillier PKE using a 3072 bit modulus N. The implementation of the threshold encryption is based on the C library *libhcs* (Tiehuis, 2018) and the Java library *Paillier Threshold Encryption Toolbox* (UTD Data and Privacy Lab, 2010).

4.4 Benchmarks

Benchmark Environment: We ran our benchmarks on a server with $2\times$ Intel Xeon Gold 6144 @ 3.5 GHz (8 physical cores) and $16 \times 32 = 512$ GB DDR4 RAM. We created 20 containers, each with 32 GB RAM and one core. The containers, running Arch Linux, were connected via a simulated WAN with a bandwidth of 100 Mbits and latency of 100 ms.

For our benchmarks, we consider the worst-case scenario for our protocol, i.e., when $k = 1$. We assume that each party holds a set of elements and the range of the elements in the union of these sets is S. Therefore, the protocol runs for $\log S$ rounds. We benchmark values of S from 10^4 to 10^{14}. We benchmark the total communication of the protocol execution for 3 to 19 parties, where each party's database has a size of 1 GB (Fig. 8b). To emulate the setting of (Tueno et al., 2020), we also evaluate the runtime for 20 to 200 parties with a single element each (Fig. 8a). The results are averaged over 10 runs for each configuration.

4.4.1 Results

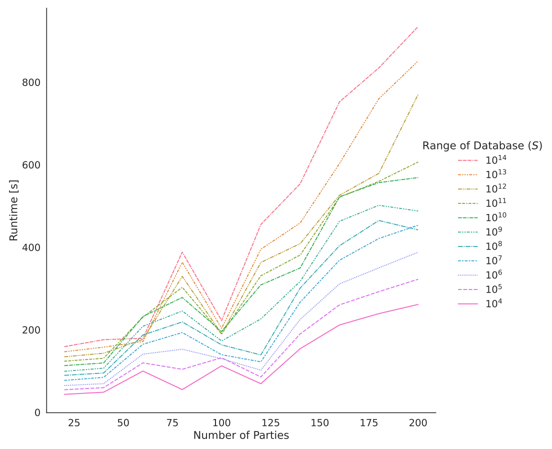
Our experimental results match with the expected asymptotic complexities. We see that the total communication scales linearly with the number of parties, and increasing the number of parties does not affect the individual communication. Moreover, the communication increases logarithmically with the range of the database.

The outliers in Fig. 8a are due to thread scheduling issues on the container executing party P_1.

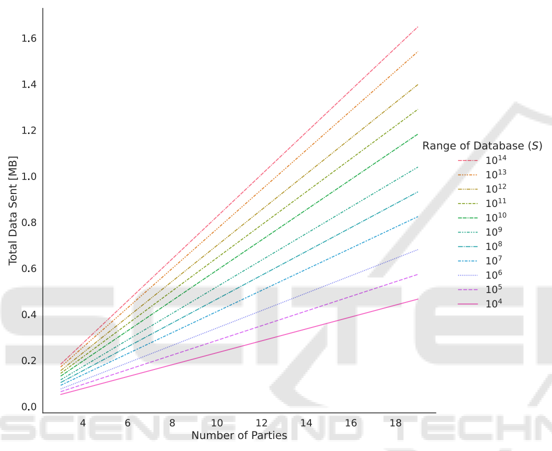
Table 3: Performance comparison of our protocol π_k^{Leaky} (Fig. 7) for computing the k^{th} ranked element among 100 parties connected via WAN with the numbers reported in (Tueno et al., 2020) based on Yao's garbled circuit (YGC) and additively homomorphic encryption (AHE). Comm. is the communication from the client to the server in MB. S is the range of elements in the database.

	(Tueno et al., 2020) (no leakage)			**This work (leaky)**	
	YGC	**AHE1**	**AHE2**	$S = 10^4$	$S = 10^{14}$
Time (s)	197	1749	441	112.6	222.2
Comm. (MB)	0.31	1.11	0.32	0.040	0.143

Comparison: We compare our results with the experimental results of the previous work in (Tueno et al., 2020). The comparison of results for 100 parties with a database range of either 10^4 or 10^{14} is given in Tab. 3. In the case of $S = 10^{14}$, our protocol reduces the client communication by more than a factor of two compared



(a) Runtime



(b) Communication

Figure 8: Experimental analysis of protocol π_k^{Leaky} (Fig. 7), which computes the k^{th} ranked element of the union of n sets, for varying number of parties n. The plots show the results for different ranges of values in the database.

to the best protocol of (Tueno et al., 2020).

5 CONCLUSION

In this paper, we have presented two multi-party protocols, one for computing a class of comparison-based functions and the second for computing the k^{th} ranked element. The protocols that we constructed have better communication complexities as compared to the previous works on these specific functions, and to generic multi-party protocols that can be used for these functions. We reduce the functions to a computation of some high-level primitives and perform the computations in a star network topology, where one party communicates with every other party to execute a series of two-party computations. We show that such a design improves the efficiency of the protocols as the communication between parties during the execution are reduced to a minimum. Our protocols have communication complexities linear in the number of parties, which makes it easily scalable.

ACKNOWLEDGEMENTS

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within ATHENE.

This project was also supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by ISF grant No. 1316/18.

REFERENCES

Aggarwal, G., Mishra, N., and Pinkas, B. (2004). Secure computation of the k th-ranked element. In *EUROCRYPT*.

Ananth, P., Badrinarayanan, S., Jain, A., Manohar, N., and Sahai, A. (2019). From FE combiners to secure MPC and back. In *TCC (1)*.

Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. (2013). More efficient oblivious transfer and extensions for faster secure computation. In *CCS*.

Beaver, D., Micali, S., and Rogaway, P. (1990). The round complexity of secure protocols (extended abstract). In *STOC*.

Ben-David, A., Nisan, N., and Pinkas, B. (2008). FairplayMP: a system for secure multi-party computation. In *CCS*.

Ben-Efraim, A., Lindell, Y., and Omri, E. (2016). Optimizing semi-honest secure multiparty computation for the internet. In *CCS*.

Boyle, E., Gilboa, N., Ishai, Y., and Nof, A. (2021). Sublinear GMW-style compiler for MPC with preprocessing. In *CRYPTO (2)*.

Cachin, C., Camenisch, J., Kilian, J., and Müller, J. (2000). One-round secure computation and secure autonomous mobile agents. In *ICALP*.

Cash, D., Jarecki, S., Jutla, C. S., Krawczyk, H., Rosu, M., and Steiner, M. (2013). Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO (1)*.

Chen, M., Hazay, C., Ishai, Y., Kashnikov, Y., Micciancio, D., Riviere, T., Shelat, A., Venkitasubramaniam, M., and Wang, R. (2021). Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In *S&P*.

Choi, S. G., Hwang, K., Katz, J., Malkin, T., and Rubenstein, D. (2012). Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In *CT-RSA*.

Choudhuri, A. R., Ciampi, M., Goyal, V., Jain, A., and Ostrovsky, R. (2020). Round optimal secure multiparty computation from minimal assumptions. In *TCC (2)*.

Couteau, G. (2016). Efficient secure comparison protocols. *IACR Cryptol. ePrint Arch.*, page 544.

Cramer, R., Damgård, I., and Nielsen, J. B. (2001). Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*.

Damgård, I., Geisler, M., and Krøigaard, M. (2007). Efficient and secure comparison for on-line auctions. In *ACISP*.

Damgård, I., Geisler, M., and Krøigaard, M. (2008). Homomorphic encryption and secure comparison. *Int. J. Appl. Cryptogr.*

Damgård, I., Jurik, M., and Nielsen, J. B. (2010). A generalization of Paillier's public-key system with applications to electronic voting. *Int. J. Inf. Sec.*

Faust, S., Hazay, C., and Venturi, D. (2013). Outsourced pattern matching. In *ICALP (2)*.

Franklin, M. K. and Haber, S. (1996). Joint encryption and message-efficient secure computation. *J. Cryptol.*

Frederiksen, T. K., Lindell, Y., Osheter, V., and Pinkas, B. (2018). Fast distributed RSA key generation for semi-honest and malicious adversaries. In *CRYPTO (2)*.

Gamal, T. E. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*.

Garay, J. A., Schoenmakers, B., and Villegas, J. (2007). Practical and secure solutions for integer comparison. In *PKC*.

Gilboa, N. (1999). Two party RSA key generation. In *CRYPTO*.

Goldreich, O., Micali, S., and Wigderson, A. (1987). How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*.

Groce, A., Rindal, P., and Rosulek, M. (2019). Cheaper private set intersection via differentially private leakage. *PoPETs*.

Hazay, C. and Lindell, Y. (2008). Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*.

Hazay, C., Mikkelsen, G. L., Rabin, T., Toft, T., and Nicolosi, A. A. (2019). Efficient RSA key generation and threshold Paillier in the two-party setting. *J. Cryptol.*

Hazay, C. and Toft, T. (2010). Computationally secure pattern matching in the presence of malicious adversaries. In *ASIACRYPT*.

Hazay, C. and Venkitasubramaniam, M. (2017). Scalable multi-party private set-intersection. In *PKC (1)*.

He, X., Machanavajjhala, A., Flynn, C. J., and Srivastava, D. (2017). Composing differential privacy and secure computation: A case study on scaling private record linkage. In *CCS*.

Inbar, R., Omri, E., and Pinkas, B. (2018). Efficient scalable multiparty private set-intersection via garbled Bloom filters. In *SCN*.

Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. (2003). Extending oblivious transfers efficiently. In *CRYPTO*.

Jarvis, R. A. (1973). On the identification of the convex hull of a finite set of points in the plane. *Inf. Process. Lett.*

Kolesnikov, V., Mohassel, P., Riva, B., and Rosulek, M. (2015). Richer efficiency/security trade-offs in 2PC. In *TCC (1)*.

Kolesnikov, V., Sadeghi, A., and Schneider, T. (2009). Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS*.

Lindell, Y. and Pinkas, B. (2004). A proof of Yao's protocol for secure two-party computation. *Electron. Colloquium Comput. Complex.*

Lindell, Y., Pinkas, B., Smart, N. P., and Yanai, A. (2015). Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO (2)*.

Lindell, Y., Smart, N. P., and Soria-Vazquez, E. (2016). More efficient constant-round multi-party computation from BMR and SHE. In *TCC (B1)*.

Mohassel, P. and Franklin, M. K. (2006). Efficiency tradeoffs for malicious two-party computation. In *PKC*.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*.

Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S. G., George, W., Keromytis, A. D., and Bellovin, S. M. (2014). Blind Seer: A scalable private DBMS. In *S&P*.

Pinkas, B., Schneider, T., and Zohner, M. (2018). Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*

Rosulek, M. and Trieu, N. (2021). Compact and malicious private set intersection for small sets. In *CCS*.

Schoppmann, P., Gascón, A., and Balle, B. (2018). Private nearest neighbors classification in federated databases. *IACR Cryptol. ePrint Arch.*

Shelat, A. and Venkitasubramaniam, M. (2015). Secure computation from millionaire. In *ASIACRYPT (1)*.

Tiehuis, M. (2018). libhcs. https://github.com/tiehuis/libhcs. Accessed: 29.11.2021.

Tueno, A., Kerschbaum, F., Katzenbeisser, S., Boev, Y., and Qureshi, M. (2020). Secure computation of the k^{th}-ranked element in a star network. In *FC*.

UTD Data and Privacy Lab (2010). Paillier threshold encryption toolbox. http://cs.utdallas.edu/dspl/cgi-bin/pailliertoolbox/index.php. Accessed: 29.11.2021.

Wang, X., Ranellucci, S., and Katz, J. (2017). Global-scale secure multiparty computation. In *CCS*.

Yao, A. C. (1982). Protocols for secure computations (extended abstract). In *FOCS*.

Yao, A. C. (1986). How to generate and exchange secrets (extended abstract). In *FOCS*.

Yasuda, M., Shimoyama, T., Kogure, J., Yokoyama, K., and Koshiba, T. (2013). Secure pattern matching using somewhat homomorphic encryption. In *CCSW*.

A Additively Homomorphic Encryption

A public key encryption (PKE) scheme is said to be additively homomorphic if for two ciphertexts $c_1 = \mathsf{Enc}_{\mathsf{pk}}(m_1;r_1)$ and $c_2 = \mathsf{Enc}_{\mathsf{pk}}(m_2;r_2)$, we can efficiently compute $\mathsf{Enc}_{\mathsf{pk}}(m_1+m_2;r)$ with an independent r and without the knowledge of the secret key sk.

Threshold PKE. In a distributed scheme, shares of the secret key are held by the parties so that the combined key remains secret. In order to decrypt, the parties use their shares to compute intermediate values, which are combined eventually to form the decrypted plaintext. Threshold encryption scheme thus comprises of two functionalities: a generate functionality to generate the shares of the secret key and share among the parties, and a decryption functionality where the parties perform the decryption together.

Homomorphic Encryption Schemes. A popular instantiation of the homomorphic encryption is the Paillier encryption scheme (Paillier, 1999), which is based on the Decisional Composite Residuosity (DCR) hardness assumption for its security. In (Gilboa, 1999; Damgård et al., 2010) threshold variant of the Paillier scheme is presented. Another instantiation of homomorphic encryption is the El Gamal scheme (Gamal, 1985). Its security is implied by the Decisional Diffie-Hellman hardness assumption.

B Security of π_k^{Leaky}

Theorem 2. (Security of π_k^{Leaky}). *Assume that* $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ *is an IND-CPA secure threshold homomorphic encryption scheme. Then the protocol (Fig. 7) securely realises $\mathcal{F}_k^{\text{Leaky}}$ in the presence of semi-honest adversaries for $n \geq 2$.*

Proof. We construct a simulator S, where S is provided with the inputs of the corrupted parties and the output of the computation. Let $\mathcal{A}$ be an adversary corrupting a subset of the parties, then two cases arise:

Case 1: $\mathcal{A}$ corrupts a strict subset $\mathcal{I}$ of the parties *excluding* P_1. Let x_k be the k^{th} ranked element in the union of all the sets. The simulator S has input D_i, $i \in \mathcal{I}$ and x_k, and is defined as follows:

1. Given $D_i,\ i \in \mathcal{I}$ and x_k, the corrupted parties are invoked by S on their corresponding inputs.
2. S generates $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ and invokes S_{Gen} for the key generation phase.
3. S plays the role of P_1 on an arbitrary set of inputs, against the corrupted parties.
4. For each iteration j, let $m_j = [a+b/2]$ be the median of the range as computed by each party. If $x_k < m_j$, then S returns 1 to all parties. If $x_k > m_j$, S returns 0 to the parties.

In this case, the view generated by the simulator is identical to the view of the honest parties in the real protocol. In the case when $x_k < m$, it implies that L (number of elements less than $m) \geq k$, then in the real execution as well as in the simulation, $b = m-1$. When $x_k > m$, it implies that G (the number of elements greater than $m) \geq N-k+1$, then in the real execution and in the simulation, $a = m+1$. When $x_k = m$, L (number of elements less than $m) \leq k-1$ and G (number of elements greater than $m) \leq N-k$, then in both executions m is the k^{th} element. Hence, the view of the honest parties in both executions are identical.

Case 2: $\mathcal{A}$ corrupts a strict subset $\mathcal{I}$ of parties including P_1. Let x_k be the k^{th} ranked element of the union of all databases D_j. Here the simulator S has input sets D_i, $i \in \mathcal{I}$ and x_k, and is defined as follows:

1. Given $D_i,\ i \in \mathcal{I}$ and x_k, the corrupted parties are invoked by the simulator on their corresponding inputs.
2. S generates $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$ and invokes the simulator $S_{\mathsf{Gen}}(\mathsf{pk})$ for π_{Gen}^{SH} in the key generation phase.
3. S plays the honest parties' role against P_1 in the protocol. For the j^{th} iteration, let $m_j = [a+b/2]$ be computed by S. The simulator sends encryptions of two arbitrary inputs to P_1 and P_1 computes two sets of sums
   (a) If $m_j < x_k$, the simulator invokes $S_{\mathsf{Dec}}(r_1)$ for the decryption of C and $S_{\mathsf{Dec}}(N-k+r_2)$ for the decryption of C', where $r_1, r_2 \in \mathbb{Z}_k$. Then P_1 returns 0 and the simulator sets $a = m_j + 1$.
   (b) If $m_j > x_k$, the simulator invokes $S_{\mathsf{Dec}}(k+r_1)$ for the decryption of C and $S_{\mathsf{Dec}}(r_2)$ for the decryption of C', where $r_1, r_2 \in \mathbb{Z}_k$. Then P_1 returns 1 and the simulator sets $b = m_j - 1$
   (c) If $m_j = x_k$, then m_j is the k^{th} element.

In this case the difference lies in the encryptions sent to P_1. In the real execution, the parties send encryptions of their input to P_1 whereas the simulator sends encryptions of arbitrary inputs to P_1. Hence, the indistinguishability of the two views follows from the privacy of the threshold homomorphic encryption scheme.

Thus, the above protocol securely computes the k^{th} ranked element of the union of n sets. ■