The Multiple Millionaires' Problem

Tamir Tassa[1] and Avishay Yanai[2]

[1] The Open University, Israel. tamirta@openu.ac.il
[2] VMware Research. ay.yanay@gmail.com

Abstract. We study a fundamental problem in Multi-Party Computation, which we call the Multiple Millionaires' Problem (MMP). Given a set of private integer inputs, the problem is to identify the subset of inputs that equal the maximum (or minimum) of that set, without revealing any further information on the inputs beyond what is implied by the desired output. Such a problem is a natural extension of the Millionaires' Problem, which is the very first Multi-Party Computation problem that was presented in Andrew Yao's seminal work [31]. A closely related problem is MaxP, in which the value of the maximum is sought. We propose several approaches towards the solution of those fundamental problems as well as concrete solutions, and compare their performance. As applications of privacy-preserving computation are more and more commonly implemented in industrial systems, MMP and MaxP become important building blocks in privacy-preserving statistics, machine learning, auctions and other domains. One of the prominent advantages of our novel protocols is their simplicity. As they solve fundamental problems that are essential building blocks in various application scenarios, we believe that our systematic study of solutions to those problems, and the comparison between them, will serve well future researchers and practitioners of secure distributed computing.

Keywords: Multi-party computation · Millionaires' problem · The maximum problem · Privacy-preserving computation.

1 Introduction

Whenever the notion of Multi-Party Computation (MPC) is mentioned, it is almost always followed by a citation of Andrew Yao's seminal work from 1982, entitled "Protocols for secure computation" [31]. Yao presented there the first MPC protocol; the problem solved by that protocol was the Millionaires' Problem (MP): given two parties, P_1 and P_2, each one holding a private nonnegative integer – x_1 and x_2, respectively, determine whether $x_1 < x_2$ or not, without revealing any further information on x_1 and x_2.[3]

As MP is an essential building block for a vast array of MPC problems, many studies have presented over the years additional solutions to it, e.g. [3,7,9,13,14,18,20,22,26,27]. We proceed to define a natural extension of MP:

[3] This problem is sometimes referred to as the Greater Than problem.

Definition 1. *Let P_n, $n \in [N] := \{1, \ldots, N\}$, be N mutually non-trusting parties that hold private integer values $x_n \geq 0$. Let $M := \max\{x_n : n \in [N]\}$. Then the Multiple Millionaires' Problem (*`MMP`*) is the MPC problem of determining all indices $n \in [N]$ for which $x_n = M$.*

We note that `MMP` treats all N inputs similarly, while `MP` does not. Hence, when $N = 2$, `MMP` coincides with `MP` only when $x_1 \neq x_2$; if, however, $x_1 = x_2$, then `MP` will issue a negative answer that implies that $x_1 \geq x_2$, while `MMP` will issue the accurate answer that $x_1 = x_2$.

A related but different MPC problem is the following.

Definition 2. *In the same setting as in Definition 1, the Maximum Problem (*`MaxP`*) is the MPC problem of computing $M = \max\{x_n : n \in [N]\}$.*

As we show later, `MMP` can be reduced to `MaxP` with an additive overhead of linear size and constant depth (see Definitions 3-4), while `MaxP` can be reduced to `MMP` with an additive constant overhead of both size and depth.

The desired output in `MMP` is the *identities* of the "richest" members in some group of N parties ("millionaires"), and the goal is to carry out that computation in a manner that prevents those parties from learning any further information on the inputs of their peers, besides what they may infer from their own input and the computed output.

Even though we focus here, for the sake of convenience, on the case of target functions that need to be maximized, our discussion includes also the case where the target function needs to be minimized. Namely, any algorithm for `MMP` can be easily used to find all indices $n \in [N]$ for which $x_n = m := \min\{x_n : n \in [N]\}$, and any algorithm for `MaxP` can be easily converted to an algorithm that computes the minimum m rather than the maximum M.

Identifying a set of parties that maximize (or minimize) some target function is a fundamental problem that is most useful in statistical analysis of data. In particular, it is an essential building block in machine learning, as many machine learning algorithms, such as K-means clustering [5,24], K-nearest neighbors [8,21], or fingerprint-matching [4,19] include the identification of a subset of points in an Euclidean space that are closest to a given point in that space. For example, in K-means clustering one is given a set of K means $\mathcal{M} := \{m_1, \ldots, m_K\} \subset \mathbb{R}^d$, and a set of observations $\mathcal{X} := \{x_1, \ldots, x_N\} \subset \mathbb{R}^d$, and for each given observation $x_n \in \mathcal{X}$, it is needed to find the mean (or means) in $\mathcal{M}$ that minimize the distance to x_n. In *privacy-preserving* machine learning, and in particular in federated (collaborative) learning, the observations need to remain secret; in such a case, all computations must be carried out in a privacy-preserving manner and, therefore, one has to solve an instance of `MMP`. In the K-nearest neighbors algorithm, on the other hand, the goal is to find the K points out of a set of N points, $\mathcal{X} = \{x_1, \ldots, x_N\} \subset \mathbb{R}^d$, that are closest to a new observation $y \in \mathbb{R}^d$; namely, the sought-after points here are the K points that issue the *smallest* values for the target function, $f(x) := \|x - y\|$, not the *minimal*, as in `MMP`. When the k-nearest neighbors algorithm has to be applied in a privacy-preserving manner, then the computational problem that it has to solve may be reduced to a sequence of instances of `MMP`.

A different application where `MMP` plays a crucial role is that of privacy-preserving (sealed-bid) auctions. There too, it is needed to identify a set of parties (bidders) that maximize a target function (the bid), without revealing any further information on the inputs (bids). See the seminal work of Naor et al. [25] and the recent survey [2] and the references therein.

The cost of privacy-preserving computation can be roughly separated into two metrics: communication and latency (being the time interval from submitting the inputs until obtaining the output). Typically, there is a tradeoff between the two, as expressed by the solutions presented in this paper (see Section 7.3). Specifically, we study the tradeoff between the size of a protocol and its depth (see Definitions 3-4). The communication-latency tradeoff is relevant in particular to GMW-style MPC protocols [16], in which the parties interact for every non-linear gate, and to leveled fully homomorphic encryption (FHE) based protocols, in which circuit evaluation is done over ciphertexts. In the former, the latency is affected mainly by the depth of the protocol, as the parties interact in every non-linear operation. In the latter, the depth has an impact on the length of the keys with which the inputs are encrypted, as well as on the computation time for every operation. Indeed, deep circuits imply larger encryption keys, what implies larger ciphertexts, which are presented as polynomials of larger degree, and, consequently, multiplying polynomials incurs larger runtimes.

1.1 Our contributions and outline

Even though `MMP` is a very natural extension of the (two-party) Millionaires' Problem, very few studies so far devised MPC protocols for solving it or other related problems (see the survey of related work in Section 8.) In this study we propose several approaches towards its solution as well as concrete solutions.

We begin with a discussion of essential preliminaries in Section 2: in Section 2.1 we present an abstraction method that will allow us to present and analyze secure computation protocols in a general context that is indifferent to the underlying setting and the implementation details; then, in Section 2.2, we present the complexity measures for assessing the performance of such protocols. In Section 3 we present a binary-tree-based protocol (Protocol 1) and a generalization to any degree (Protocol 4); these protocols are based on a "tournament" between the "candidates" (inputs) to determine the "winners" (maximal inputs), through several rounds of comparisons. To the best of our knowledge, the general-degree tree-based protocol (Protocol 4) is new, whereas the binary-tree based-protocol was already proposed and used in the literature, e.g. [12,24]. In Section 4 we present novel constant-depth protocols for solving `MaxP` and `MMP` (Protocols 5 and 6) that are tailored to cases where the inputs are drawn from a small domain. Then, we present in Section 5 protocols that are based on the inputs' decomposition into bits (Protocol 7) or into digits in larger bases (Protocol 8). In Section 6 we present a protocol for solving `MaxP` and `MMP` that is inspired by the protocols of [1] for finding the k-th ranked element in a union of private datasets (namely, the value in the k-th entry in the unified dataset when it is sorted in an increasing order). We evaluate the performance of all of those protocols in Section 7; the analysis is based on an implementation of passively secure MPC protocols, which allows ignoring 'noisy' costs, like consistency checks of inputs, that are typically added to actively secure protocols. Furthermore, the asymptotic analysis of the cost of passively secure protocols capture the actively secure ones as well, as there are efficient compilers from passive to active security (e.g., [15]). We review related work in Section 8 and conclude in Section 9.

A prominent advantage of our novel protocols is their simplicity. Since `MaxP` and `MMP` are essential building blocks in important applications, our systematic study and comparison of solutions to those problems will serve well future researchers of MPC and practitioners of secure distributed computing.

We conclude this introduction with a summary of the costs of the various `MMP` protocols that we present here, see Table 1. Some of the protocols assume that the inputs are given as field elements, while others assume that they are given by their bits or digits. However, the form of the inputs does not affect the asymptotic analysis since there are constant-depth bit-decomposition protocols (e.g., [30]) that can be invoked prior to the execution of the protocols that rely on bit-decomposed inputs, and that preprocessing stage incurs a constant additive overhead.

	size	preDepth	onDepth
Protocol 1 [12,24]	$O(NB + NK)$	1	$5 \cdot \lceil \log N \rceil + 4$
Protocol 4	$O(N'NB + N'NK)$	3	$3 + 12\lceil \log_{N'} N \rceil$
Protocol 6	$(2^B NK)$	3	9
Protocol 7	$O(NBK)$	3	$2 + 6B$
Protocol 8	$O(\frac{NBK2^d}{d})$	3	$2 + 11B/d$
Protocol 11 [1]	$O(NB^2 + NBK)$	3	$8B + 4$

Table 1. Analysis of the size, preprocessing depth and online depth of the protocols presented in this paper, according to Definitions 3-4 (see Section 2.2), when instantiated in a setting of honest majority, semi-honest adversary and statistical security (See Section 7 for details). The size is given asymptotically, while the preprocessing depth and online depth are given exatcly. The parameters are: K is the number of parties; N is the number of inputs; each input is drawn from the domain $[0, 2^B - 1]$; $N' \in [2, N]$ is the degree of the general comparison tree in Protocol 4; d is the bit-length of digits in Protocol 8.

2 Preliminaries

2.1 The Arithmetic Black-Box (ABB)

Practical protocols for secure computation use a circuit representation of the functionality that they compute. The gates in the circuit may be arithmetic, where computation is carried out by addition and multiplication gates over an agreed upon field, or Boolean, where computation is carried out by AND and XOR gates. Note that Boolean gates adhere to arithmetics over $\mathbb{Z}_2$, thus, they can use the arithmetic notation ('$+$', '$\cdot$') as well.

Typically, privacy-preserving protocols abstract out the underlying techniques through an abstraction called an "Arithmetic Black Box" (ABB). An ABB is an ideal functionality that can be realized in many ways, depending on the setting (e.g., network topology and synchrony assumptions) and the security needs (i.e., semi-honest/malicious, static/adaptive adversaries, with/without abort, perfect/statistical/computational security, and different collusion scenarios). The privacy-preserving protocol directly inherits these properties from the sub-protocol that realizes the ABB. This way, it is possible to describe a privacy-preserving protocol that can fit several settings at the same time, given a suitable realization of the ABB functionality in the protocol. Furthermore, by relying on the ABB functionality, one does not have to analyse the communication and computation complexity of its operations, as they depend on the specific realization. The ABB ideal functionality is given in Functionality 1. Abstraction using an ideal functionality is a common practice in the art of secure protocol design, which enables focusing on the logic of the application rather than on the underlying infrastructure.

In the beginning of the protocol the parties initialize the ABB functionality with the field $\mathbb{F}$ over which all computations take place. We assume hereinafter that $\mathbb{F} = \mathbb{Z}_p$ for a sufficiently large prime p that is greater than all inputs of the `MMP` problem. After that initialization the parties may use any interface listed in Functionality 1 as many times as needed, as part of the privacy-preserving protocol. See [6] for the syntax and usability of ideal functionalities and [10] for the introduction of the arithmetic black-box.

In Functionality 1 we use the term *handle*. A handle can be seen as a public pointer to the secret held by the ABB, so that the parties can relate to it in their calls. A possible instantiation of a handle is as follows. On inputting a secret or a public value x, the ABB picks a unique identifier (e.g., a long random string) and outputs it as x's handle. In addition, when the parties invoke a computation on the secrets represented by handles $h_1, \ldots, h_k$ the ABB performs the computation on the secrets and picks a new unique identifier for the result.

In a protocol description, instead of explicitly mentioning that the parties send some message through the ABB interface, it is convenient to use a shorthand. Specifically:

– instead of writing that party P_n calls $(\mathsf{PrivateInput}, n, a)$ and all other parties call $(\mathsf{PrivateInput}, n)$ so that all parties obtain $[a]$, we write simply $[a] \leftarrow (\mathsf{PrivateInput}, n, a)$.

– instead of writing that all parties call $(\mathsf{Add}, [a], [b])$ and obtain the handle $[c]$', we write $[c] \leftarrow [a] + [b]$;
– a call to $(\mathsf{Multiply}, [a], [b])$ is replaced with $[c] \leftarrow [a] \cdot [b]$;
– a call to $(\mathsf{Duplicate}, [a])$ which returns to all parties a handle $[b]$ with the same value as $[a]$ is replaced with $[b] \leftarrow [a]$;
– a call to $(\mathsf{AffineComb}, [a_1], \ldots, [a_k], c_0, c_1, \ldots, c_k)$ is replaced with $[c] \leftarrow c_0 + c_1 \cdot [a_1] + \cdots + c_k \cdot [a_k]$;
– a call to $(\mathsf{Inverse}, [a])$ is replaced with $[c] \leftarrow [a^{-1}]$;
– a call to $(\mathsf{Compare}, [a], [b])$ is replaced with the handle that it returns, i.e. $[1_{a<b}]$.[4]
– a call to $(\mathsf{Equal}, [a], [b])$ is replaced with the result $1_{a=b}$.
– a call to $(\mathsf{OR}_m, [a_1], \ldots, [a_m])$ is replaced with $[c] \leftarrow \bigvee_{i=1}^{m} [a_i]$.

There are plenty of potential instantiations for the ABB functionality, depending on the number of parties (e.g., it is common to have more efficient protocols for a small number of parties, say $N \in \{2, 3, 4, 5\}$), the adversarial model (e.g., protocols that protect against a semi-honest and static adversary who can corrupt only a few parties are obviously simpler and more efficient than those protecting against an active and adaptive adversary who may corrupt almost all parties), the network topology (e.g., it is easier to communicate over a full mesh rather than over a sparse network), and the underlying setting.

We conclude by noting that some of the interfaces in the ABB functionality could be realized by other ones; see more on that in Appendix A.1.

2.2 Measures of protocols' complexity

Hereinafter, when analyzing the complexity of a given MPC protocol we will refer to the protocol's size and depth, where the *size* addresses the protocol's communication complexity and the *depth* addresses its round complexity. Each MPC protocol induces a tree of calls to the ABB functionality. Let $\mathcal{I}$ be the set of all interfaces in the ABB Functionality 1. Then each node in the tree that is induced by a given protocol is labeled by some $\mathsf{A} \in \mathcal{I}$. All nodes in the same distance from the root node represent calls that can be executed in parallel. On the other hand, a pair of a father node with a son node represents a pair of calls to the ABB functionality where the latter call depends on the completion of the former.

Definition 3 (Protocol's size). *Given an MPC protocol Π we let c_{A} be the number of invocations of the intereface $\mathsf{A} \in \mathcal{I}$ during Π's execution. Then the size of protocol Π is defined as $\mathsf{size}(\Pi) = \sum_{\mathsf{A} \in \mathcal{I}} c_{\mathsf{A}} \cdot \mathsf{size}(\mathsf{A})$, where $\mathsf{size}(\mathsf{A})$ is the message complexity of the invocation of A in a specific implementation of the ABB functionality.*

Definition 4 (Protocol's depth). *Let $w := (\mathsf{A}_1, \ldots, \mathsf{A}_d)$ be a path in the tree that protocol Π induces. Its depth is defined as $\mathsf{depth}_w := \sum_{i=1}^{d} \mathsf{depth}(\mathsf{A}_i)$,*

[4] If $\mathcal{P}$ is a predicate then $1_{\mathcal{P}}$ is a bit that equals 1 if the predicate holds and 0 otherwise.

FUNCTIONALITY 1 (*Arithmetic Black-Box*)

Initialize. On input $(\mathsf{Init}, \mathbb{F})$ from all parties, store the field $\mathbb{F}$.

Public Input. [PuI] On input $(\mathsf{PublicInput}, a)$ from all parties, where $a \in \mathbb{F}$, store a and return a handle $[a]$ to the parties.

Private Input. [PrI] On input $(\mathsf{PrivateInput}, n, a)$ from party P_n, where $a \in \mathbb{F}$, and input $(\mathsf{PrivateInput}, n)$ from all other parties, store a and return a handle $[a]$ to the parties.

Public Output. [PuO] On input $(\mathsf{PublicOutput}, [a])$ from all parties, reveal a to all parties.

Private Output. [PrO] On input $(\mathsf{PrivateOutput}, n, [a])$ from all parties, reveal a to party P_n.

Random. [Rnd] On input (Random) from all parties, sample $r \leftarrow \mathbb{F}$ uniformly and return the handle $[r]$ to the parties.

Addition. On input $(\mathsf{Add}, [a], [b])$ from all parties, compute $c = a + b$ in $\mathbb{F}$ and return a handle $[c]$ to all parties.

Multiplication. [Mul] On input $(\mathsf{Multiply}, [a], [b])$ from all parties, compute $c = a \cdot b$ in $\mathbb{F}$ and return a handle $[c]$ to all parties.

Duplicate. On input $(\mathsf{Duplicate}, [a])$ from all parties, the functionality generates a new handle $[b]$ to a secret b where $b = a$, and returns the handle $[b]$ to all parties.

Affine combination. On input $(\mathsf{AffineComb}, [a_1], \dots, [a_k], c_0, c_1, \dots, c_k)$ from all parties, compute $c = c_0 + \sum_{i=1}^{k} c_i a_i$ in $\mathbb{F}$ and return a handle $[c]$ to all parties.

Inverse. [Inv] On input $(\mathsf{Inverse}, [a])$ from all parties, where $a \in \mathbb{F} \setminus \{0\}$, compute $c = a^{-1}$ and return a handle $[c]$ to all parties.

Equality. [Equ] On input $(\mathsf{Equal}, [a], b)$ from all parties, compute and return $c \leftarrow 1_{a=b}$.

Comparison. [Com] On input $(\mathsf{Compare}, [a], [b])$ from all parties, compute $c \leftarrow 1_{a<b}$ and return a handle $[c]$ to all parties. (We focus here on prime-ordered fields $\mathbb{F} = \mathbb{Z}_p$ in which $a < b$ means that a is smaller than b when both a and b are viewed as integers.)

Or. [Or] On input $(\mathsf{OR}_m, [a_1], \dots, [a_m])$ from all parties, if $a_i \in \{0, 1\}$ for all $i \in [m]$, compute $c \leftarrow \bigvee_{i=1}^{m} a_i$ and return a handle $[c]$ to all parties; otherwise, return $\perp$ to all parties.

where $\mathsf{depth}(\mathsf{A}_i)$ *is the round complexity of the invocation of* A_i *in a specific implementation of the ABB functionality. Then the depth of protocol* Π *is defined as* $\max_w \mathsf{depth}_w$ *where the maximum is over all paths in the tree that* Π *induces.*

In our analysis, we break the depth of the implementation of each interface A to its part that is independent of the input and its part that does depend on the input. For an interface A, we call the former the *preprocessing depth* and denote it by $\mathsf{preDepth}(\mathsf{A})$, while the latter is called the *online depth* and is denoted $\mathsf{onDepth}(\mathsf{A})$. Similarly, the depth of a protocol Π is also split into a preprocessing depth and an online depth, $\mathsf{depth}(\Pi) = \mathsf{preDepth}(\Pi) + \mathsf{onDepth}(\Pi)$. Note that if protocol Π invokes a set of interfaces $\mathcal{J} \subseteq \mathcal{I}$, then $\mathsf{preDepth}(\Pi) = \max\{\mathsf{preDepth}(\mathsf{A}) : \mathsf{A} \in \mathcal{J}\}$ because the preprocessing part for all interface invocations can be done simultaneously. As for the online depth, the exact depth depends on the invocation pattern in each protocol.

When analyzing the size and depth of our protocols, we use abbreviated notations as listed in Functionality 1. So, for example, the notation $[\textsc{Com}]$ denotes the *size* of the Compare interface, when speaking about the size of a protocol, while it denotes the *preprocessing depth* (resp., *online depth*) of the interface when discussing the protocol's preprocessing depth (resp., online depth). In such analysis we ignore calls to Add, AffineComb and Duplicate, since they incur no communication at all.

3 Tree-based protocols

In this section we present protocols that solve `MaxP` (i.e., finding the maximum of all inputs) by decomposing the original problem, that involves many inputs, into a hierarchy (tree) of smaller instances of `MaxP`, each one consisting of a smaller number of inputs. In Section 3.1 we describe a protocol that is based on a binary tree; in that protocol, each smaller instance of `MaxP` involves just two inputs. Then, in Section 3.2, we present a generalized protocol in which the smaller `MaxP` instances involve up to N' (where $2 \leq N' \leq N$) inputs.

3.1 A binary tree-based protocol

Here we present a basic `MMP` protocol that reduces the Multiple Millionaires' Problem into the (Two) Millionaires' Problem `MP`. Protocol 1 is executed by the parties $P_1, \ldots, P_N$ towards identifying all maximal inputs. The protocol uses a binary tree based comparison in order to compute the maximal value.

Initially, all parties input their secret inputs, $x_1, \ldots, x_N$ (Lines 1-2). The computation that follows is performed in L rounds, where $L = \lceil \log N \rceil$ (hereinafter, unless otherwise stated, $\log = \log_2$). There are N_ℓ candidates in the ℓ-th round ($\ell \in \{1, \ldots, L\}$), denoted $x_1^\ell, \ldots, x_{N_\ell}^\ell$. In the first round the candidates are $x_n^1 = x_n$, $n \in [N]$, so N_1 is initialized to N (Line 3).

As mentioned, the computation of the maximal value M is done in L rounds (Lines 4-10). In the ℓ-th round every two consecutive candidates, x_k^ℓ and x_{k+1}^ℓ, are compared (Line 6). The comparison result is used in order to compute the maximum of those two candidates using the equality $\max\{a, b\} = a + 1_{a<b} \cdot (b - a)$; it is then stored in $[x_{(k+1)/2}^{\ell+1}]$ and percolated to the next round (Line 7). If the number of candidates in the ℓ-th round, N_ℓ, is odd, then the last candidate $(x_{N_\ell}^\ell)$ is directly percolated to the next round (Lines 8-9). Finally, the number of values in the next layer is updated (Line 10). After completing the L rounds, the maximum M is the single value in the $(L+1)$-th layer (Line 11). The protocol ends with a testing the equality of each of the inputs to the computed maximum (Lins 12-13) and outputting the indices of all maximal inputs.

The protocol consists of N calls to PrivateInput (Line 2), followed by $N - 1$ calls to Compare and Multiply (Line 7). Afterwards, the protocol performs N calls to Equality (Line 13) and PublicOutput. Note that all calls to PrivateInput

Protocol 1: A binary-tree-based protocol for solving `MMP`

```
    Parameters: $N$ - number of inputs; $L = \lceil \log N \rceil$.
    Private Inputs: $x_1, \dots, x_N \in \mathbb{F}$.
1   forall $n \in [N]$ do
2       $[x_n^1] \leftarrow (\mathsf{PrivateInput}, n, x_n)$
3   $N_1 \leftarrow N$.
4   forall $\ell = 1, 2, \dots, L$ do
5       forall Odd $k$ s.t. $k < N_\ell$ do
6           The parties compute $[1_{x_k^\ell < x_{k+1}^\ell}]$.
7           $[x_{(k+1)/2}^{\ell+1}] \leftarrow [x_k^\ell] + [1_{x_k^\ell < x_{k+1}^\ell}] \cdot ([x_{k+1}^\ell] - [x_k^\ell])$.
8       if $N_\ell$ is odd then
9           $[x_{(N_\ell+1)/2}^{\ell+1}] \leftarrow [x_{N_\ell}^\ell]$.
10      $N_{\ell+1} \leftarrow \lceil N_\ell/2 \rceil$.
11  $[M] \leftarrow [x_1^{L+1}]$.
12  forall $n \in [N]$ do
13      The parties compute $1_{x_n^1 = M}$.
    Output: The indices $n \in [N]$ for which $x_n = M := \max\{x_i : i \in [N]\}$.
```

occur in parallel, as well as the calls to Equality and PublicOutput. In addition, all calls to Compare and Multiply occur in $\lceil \log N \rceil$ iterations. Therefore, the size and depth of Protocol 1 are:

$$\mathsf{size}(\text{Protocol 1}) = N\cdot[\textsc{PrI}]+(N-1)\cdot[\textsc{Com}]+(N-1)\cdot[\textsc{Mul}]+N\cdot[\textsc{Equ}]+N\cdot[\textsc{PuO}]\,,$$

$$\mathsf{preDepth}(\text{Protocol 1}) = \max([\textsc{PrI}], [\textsc{Com}], [\textsc{Mul}], [\textsc{Equ}], [\textsc{PuO}])\,,$$

$$\mathsf{onDepth}(\text{Protocol 1}) = [\textsc{PrI}] + \lceil \log N \rceil \cdot ([\textsc{Com}] + [\textsc{Mul}]) + [\textsc{Equ}] + [\textsc{PuO}]\,.$$

3.2 A protocol based on higher degree trees

Here we present Protocol 4 for solving `MMP` and `MaxP`. That protocol is also based on a comparison tree, only that here the degrees of all nodes are (at most) some parameter $N' \geq 2$. The depth of Protocol 4 is $\lceil \log_{N'} N \rceil$. If we take $N' = 2$ we restore Protocol 1. If, on the other hand, we take $N' = \sqrt[O(1)]{N}$, we get a shallow tree with a depth of $O(1)$.

Before presenting Protocol 4, we present two sub-protocols that it invokes, for solving `MMP` and `MaxP` in constant depth. Sub-protocol 2 securely computes the indices of the maximal value. It receives as input handles to N' candidate values. First, it computes in parallel the comparison bits between every pair of inputs (Line 1-2). It then uses those bits to compute a bit for each input that indicates whether that input is maximal (Lines 3-4).

Sub-protocol 3 computes the maximal value. It too receives as input handles to N' candidate values. It then invokes Sub-protocol 2 and gets hanldes to the bits $[b_i]$, $i \in [N']$, that inidcate the maximality of each input (Line 1). It then proceeds to add up all maximal inputs into $\hat{M}$ and count them into $\hat{N}$ (Lines 2-3). The maximum is obtained by multiplying $\hat{M}$ with the inverse of $\hat{N}$.

Sub-protocol 2: An all-to-all-comparison protocol for solving MMP

```
   Inputs: Handles $[x_1], \dots, [x_{N'}]$ where $x_i \in \mathbb{F}$.
1  forall $1 \leq i \neq j \leq N'$ do
2      $[b_{i,j}] \leftarrow [1_{x_i < x_j}]$.
3  forall $i \in [N']$ do
4      $[b_i] \leftarrow 1 - \bigvee_{j \in [N], j \neq i} [b_{i,j}]$.
   Output: The handles $([b_1], \dots, [b_{N'}])$.
```

Sub-protocol 3: A sub-protocol for solving MaxP

```
   Inputs: Handles $[x_1], \dots, [x_{N'}]$ where $x_i \in \mathbb{F}$.
1  $([b_1], \dots, [b_{N'}]) \leftarrow$ MMP$([x_1], \dots, [x_{N'}])$.
2  $[\hat{M}] \leftarrow \sum_{i \in [N']} ([b_i] \cdot [x_i])$.
3  $[\hat{N}] \leftarrow \sum_{i \in [N']} [b_i]$.
4  $[M] \leftarrow [\hat{N}^{-1}] \cdot [\hat{M}]$.
   Output: The handle $[M]$ to $M = \max\{x_i : i \in [N']\}$.
```

We now turn to Protocol 4. Lines 1-3 in it are the same as in Protocol 1. Lines 4-7 are equivalent to Lines 4-10 in Protocol 1, with the only difference being the fact that in the ℓ-th iteration, $1 \leq \ell \leq \lceil \log_{N'} N \rceil$, the N_ℓ candidates are grouped into groups of at most N' candidates in each (instead of 2 in Protocol 1) and then the maximum of each such group is percolated to the next iteration. The conclusion of the protocol (Lines 8-10) is similar to the conclusion of Protocol 1 (Lines 11-13 there).

Protocol 4: A general-tree-based protocol for solving MMP

```
    Parameters: $N$ - number of inputs; $N'$, where $2 \leq N' \leq N$ - number of
                inputs that are compared in one call to Sub-protocol 3 (MaxP);
                $L' = \lceil \log_{N'} N \rceil$.
    Private Inputs: $x_1, \dots, x_N \in \mathbb{F}$.
1   forall $n \in [N]$ do
2       $[x_n^1] \leftarrow (\mathsf{PrivateInput}, n, x_n)$
3   $N_1 \leftarrow N$.
4   forall $\ell = 1, 2, \dots, L'$ do
5       forall $k$ s.t. $k = 1 \mod N'$ and $k \leq N_\ell$ do
6           $[x_{1+(k-1)/N'}^{\ell+1}] \leftarrow$ MaxP$([x_k^\ell], [x_{k+1}^\ell], \dots, x_{\min\{k+N'-1, N_\ell\}}^\ell)$.
7       $N_{\ell+1} \leftarrow \lceil N_\ell / N' \rceil$.
8   $[M] \leftarrow [x_1^{L'+1}]$.
9   forall $n \in [N]$ do
10      The parties compute $1_{x_n^1 = M}$. (Recall that $x_n^1$ is the $n$-th input).
    Output: The indices $n \in [N]$ for which $x_n = M := \max\{x_i : i \in [N]\}$.
```

We conclude with analyzing the costs of those protocols. Sub-protocol 2's size is $((N')^2 - N') \cdot [\textsc{Com}] + N' \cdot [\textsc{Or}]_{N'}$, where the subscript N' in $[\textsc{Or}]_{N'}$ denotes the number of operands in the $\textsc{Or}$ operation. Sub-protocol 2's preprocessing and online depths are $\max([\textsc{Com}], [\textsc{Or}]_{N'})$ and $[\textsc{Com}] + [\textsc{Or}]_{N'}$, respectively. These costs of Sub-protocol 2 imply that the costs of Sub-protocol 3 are:

$$\mathsf{size}(\text{Sub-protocol 3}) = ((N')^2 - N') \cdot [\textsc{Com}] + N' \cdot [\textsc{Or}]_{N'} + (N'+1) \cdot [\textsc{Mul}] + [\textsc{Inv}]\,,$$

$$\mathsf{preDepth}(\text{Sub-protocol 3}) = \max([\textsc{Com}], [\textsc{Or}]_{N'}, [\textsc{Mul}], [\textsc{Inv}])\,,$$

$$\mathsf{onDepth}(\text{Sub-protocol 3}) = [\textsc{Com}] + [\textsc{Or}]_{N'} + 2[\textsc{Mul}] + [\textsc{Inv}]\,.$$

Letting $[\textsc{MaxP}]$ denote the costs of Sub-protocol 3, Protocol 4's size is

$$\mathsf{size}(\text{Protocol 4}) = N \cdot [\textsc{PrI}] + T(N, N') \cdot [\textsc{MaxP}] + N \cdot [\textsc{Equ}]\,,$$

where $T(N, N') = \sum_{\ell=1}^{L'} \lceil N_\ell / N' \rceil$ and N_ℓ is defined recursively in Lines 3 & 7. (Note that if $N = (N')^{L'}$ the tree is regular, in the sense that all non-leaf nodes have an out-degree N', and then $T(N, N') = \frac{N-1}{N'-1}$.) Protocol 4's depths are:

$$\mathsf{preDepth}(\text{Protocol 4}) = \max([\textsc{PrI}], [\textsc{MaxP}], [\textsc{Equ}])\,,$$

$$\mathsf{onDepth}(\text{Protocol 4}) = [\textsc{PrI}] + L' \cdot [\textsc{MaxP}] + [\textsc{Equ}]\,.$$

4 Constant-depth protocols for small domains

We begin this section by presenting a constant-depth protocol for solving `MaxP` (Section 4.1). Then we use that protocol as a basis for a constant-depth protocol for solving `MMP` (Section 4.2). The size of the protocols is proportional to the underlying field size and, therefore, they are relevant in cases where there is a small known upper bound on the inputs.

4.1 A protocol for solving `MaxP`

Protocol 5 starts with each P_n, $n \in [N]$, submitting its input, which is an integer $x_n \in [0, Q)$, for some publicly known upper bound Q, in the form of $Q - 1$ bits $y_{n,i}$ that represent it (Lines 1-3). Specifically, each input $x \in [0, Q)$ is represented by a sequence of $Q - 1$ bits in which the first x bits are 1 and the proceeding ones are all 0; it is that form of input encoding that restricts the scope of this protocol to small domains. Then, the parties compute a similar representation of M into the vector $\mathbf{z} = (z_1, \ldots, z_{Q-1})$, using the $\bigvee$ operator (Lines 4-5). The $\bigvee$ operator can be computed by a constant-depth sub-protocol (see Appendix A.2.3). Finally, the maximum M is computed by adding all entries in $\mathbf{z}$ (Line 6).

The protocol performs $N(Q - 1)$ parallel invocations of $\mathsf{PrivateInput}$ (Lines 1-3) and $Q - 1$ parallel invocations of Or (Lines 4-5). Hence, its costs are:

$$\mathsf{size}(\text{Protocol 5}) = N(Q-1) \cdot [\textsc{PrI}] + (Q-1) \cdot [\textsc{Or}]_N\,,$$

$$\mathsf{preDepth}(\text{Protocol 5}) = \max([\textsc{PrI}], [\textsc{Or}]_N)\,, \quad \mathsf{onDepth}(\text{Protocol 5}) = [\textsc{PrI}] + [\textsc{Or}]_N\,.$$

Protocol 5: A monotone-representation-based protocol for solving MaxP

```
   Parameters: $N$ - number of inputs; $Q$ - an upper bound on the inputs (each
               input is in $[0, Q-1)$.
   Private Inputs: $x_1, \dots, x_N \in [0, Q)$.
1  forall $n \in [N]$ do
2      forall $i = 1, \dots, Q-1$ do
3          $[y_{n,i}] \leftarrow (\mathsf{PrivateInput}, n, y_{n,i})$, where $y_{n,i} = 1_{x_n \geq i}$
4  forall $i = 1, \dots, Q-1$ do
5      $[z_i] \leftarrow \bigvee_{n \in [N]} [y_{n,i}]$
6  $[M] \leftarrow \sum_{i=1}^{Q-1} [z_i]$
   Output: A handle $[M]$ to $M := \max\{x_n : n \in [N]\}$.
```

4.2 A protocol for solving `MMP`

Here we present Protocol 6 – an `MMP` protocol that is based on the `MaxP` Protocol 5. It starts with an invocation of Protocol 5 to compute the maximum M of all inputs (Line 1). Then, the parties go over all inputs and test their equality to M (Lines 2-3). The protocol's output identifies all inputs that equal M.

Protocol 6: A monotone-representation-based protocol for solving MMP

```
   Parameters: $N$, $Q$ - two positive integers, denoting the number of inputs and
               an upper bound on them.
   Private Inputs: $x_1, \dots, x_N \in [0, Q)$;
1  The parties compute $[M] \leftarrow [\mathtt{MaxP}\{x_1, \dots, x_N\}]$.
2  forall $n \in [N]$ do
3      The parties compute $1_{x_n = M}$.
   Output: The indices $n \in [N]$ for which $x_n = M := \max\{x_i : i \in [N]\}$.
```

Protocol 6 consists of a call to Protocol 5 to compute M, followed by N parallel procedures of testing the equality of each input to M (Lines 2-5). Hence, its costs are as follows:

$$\mathsf{size}(\text{Protocol 6}) = \mathsf{size}(\text{Protocol 5}) + N \cdot [\text{Equ}] + N \cdot [\text{PuO}]\,,$$

$$\mathsf{preDepth}(\text{Protocol 6}) = \max(\mathsf{preDepth}(\text{Protocol 5}), [\text{Equ}], [\text{PuO}])\,,$$

$$\mathsf{onDepth}(\text{Protocol 6}) = \mathsf{onDepth}(\text{Protocol 5}) + [\text{Equ}] + [\text{PuO}]\,.$$

By plugging the size and the preprocessing and online depths of Protocol 5 as derived in Section 4.1, we arrive at the final size and depths of Protocol 6:

$$\mathsf{size}(\text{Protocol 6}) = N(Q-1) \cdot [\text{Pri}] + (Q-1) \cdot [\text{Or}]_N + N \cdot [\text{Equ}] + N \cdot [\text{PuO}]\,;$$

$$\mathsf{preDepth}(\text{Protocol 6}) = \max([\text{Pri}], [\text{Or}]_N, [\text{Equ}], [\text{PuO}])\,,$$

$$\mathsf{onDepth}(\text{Protocol 6}) = [\text{Pri}] + [\text{Or}]_N + [\text{Equ}] + [\text{PuO}]\,.$$

5 Protocols based on the inputs' digit decomposition

Here we present two protocols for solving `MMP` that are based on the digits of the inputs' representation in some number base. We begin with Protocol 7 (Section 5.1) that uses a binary representation of the inputs. We then present Protocol 8 (Section 5.2) that reduces depth by considering a representation of the inputs in a 2^d-base, for some $d > 1$.

5.1 Binary representation of inputs

Protocol 7 gets as inputs N integers, $x_1, \ldots, x_N$. As stated earlier, we assume that all inputs are smaller than some prime p, where the underlying field is $\mathbb{F} = \mathbb{Z}_p$. Hereinafter we let $B = \lceil \log p \rceil$ denote the number of bits in the binary representation of elements in $\mathbb{F}$. Hence, the input x_n, $n \in [N]$, has a binary representation by B bits, $(x_{n,B-1}, \ldots, x_{n,0})$, meaning that $x_{n,b} \in \{0,1\}$ and $x_n = \sum_{b=0}^{B-1} x_{n,b} 2^b$. In addition, we define $x_{n,B} = 1$ for all $n \in [N]$. The protocol outputs an N-dimensional Boolean vector that identifies the maximal inputs.

The protocol starts by submitting all input bits (Lines 1-3). It then iterates over the input bits from bit number $b = B-1$, the MSB, to bit number $b = 0$, the LSB (Lines 4-7). We refer to the collection of all bits in the same bit position, $\{x_{n,b} : n \in [N]\}$, as the b-th bit column. For each of those bit columns the protocol computes $s_b := \bigvee_{n \in [N]} (x_{n,b+1} \cdot x_{n,b})$ (Line 5). If $s_b = 1$ then there is at least one input x_n for which $x_{n,b} = 1$ and, in addition, the current value of $x_{n,b+1}$ (after it may have been updated in the previous iteration) also equals 1; in that case, the protocol updates each entry in that column to equal the product of that entry and the preceding one. Otherwise, if $s_b = 0$, the protocol updates the b-th bit column to be the same as the preceding bit column. This computation is done in Lines 6-7. (Note that all computations are done in $\mathbb{Z}_2$ so that the minus and plus operations are just XORs). At the conclusion of this loop, the protocol outputs the bits in the 0-th bit column (Line 8) since those bits identify all maximal inputs, as we claim next:

Lemma 1. $x_{n,0} = 1$ *if and only if* $x_n = M = \max\{x_i : i \in [N]\}$.

Proof. For each $b = B-1, \ldots, 0$ define $x_n|^b = \sum_{i=b}^{B-1} x_{n,i} 2^i$; namely, $x_n|^b$ is the value that is obtained from x_n by zeroing its b least significant bits, $x_{n,b-1}, \ldots, x_{n,0}$. We prove, by induction on $b = B-1, \ldots, 0$, that at the completion of the protocol $x_{n,b} = 1$ iff $x_n|^b = \max\{x_i|^b : i \in [N]\}$. The case $b = 0$ coincides with the statement of the lemma since $x_n|^0 = x_n$.

The statement for the base case $b = B-1$ is easily proven. There are two cases to consider:

– If $s_b = 1$ then at least one of the inputs has a 1-bit in this column and, therefore, $\max\{x_i|^{B-1} : i \in [N]\} = 1$. Hence, after updating the shares in $x_{n,B-1}$ in Line 7, $x_{n,B-1}$ equals the product between $x_{n,B} = 1$ and the original value of $x_{n,B-1}$. That means that $x_{n,B-1}$ remains unchanged in this case and, consequently, $x_{n,B-1} = 1$ iff $x_n|^{B-1} = \max\{x_i|^{B-1} : i \in [N]\}$.

– If $s_b = 0$ then, because $x_{n,B} = 1$ for all n, all inputs have a 0-bit in this column and, therefore, $\max\{x_i|^{B-1} : i \in [N]\} = 0$. Hence, after updating the shares in $x_{n,B-1}$ in Line 7, $x_{n,B-1}$ would equal $x_{n,B} = 1$, for all $n \in [N]$. Indeed, in this case, as all bits in that column are zero, then $x_n|^{B-1} = \max\{x_i|^{B-1} : i \in [N]\}$ for all $n \in [N]$.

We proceed by induction to prove our claim for $b = B-2, \dots, 0$. Namely, we assume that $x_n|^{b+1} = \max\{x_i|^{b+1} : i \in [N]\}$, and then prove that $x_n|^b = \max\{x_i|^b : i \in [N]\}$. Here too we distinguish between two cases:

– If $s_b = 0$ then all inputs have a 0-bit in this column or in the preceding column (that was already updated in the previous iteration). We claim that in this case $\arg\max_n\{x_n|^b\} = \arg\max_n\{x_n|^{b+1}\}$; namely, that all inputs that maximize $x_n|^{b+1}$ are exactly those that maximize $x_n|^b$. Indeed, if $i \notin \arg\max_n\{x_n|^{b+1}\}$ then clearly $i \notin \arg\max_n\{x_n|^b\}$. If, on the other hand, $i \in \arg\max_n\{x_n|^{b+1}\}$ then, by the induction hypothesis, $x_{i,b+1} = 1$ and therefore, because $s_b = 0$, $x_{i,b} = 0$. Hence, for all $x_i \in \arg\max_n\{x_n|^{b+1}\}$ we have $x_i|^{b+1} = x_i|^b$. Therefore, all inputs that maximize $x_n|^{b+1}$ also maximize $x_n|^b$. Hence, since the computation in Line 7 updates the shares in $x_{n,b}$ so that it equals $x_{n,b+1}$, the a-posteriori values of the bits $x_{n,b}$ identify the inputs that maximize $x_n|^b$.
– If $s_b = 1$ then at least one of the inputs that maximized $x_n|^{b+1}$ has a 1-bit in the b-th column. Therefore, $\arg\max_n\{x_n|^b\}$ consists exactly of all inputs in $\arg\max_n\{x_n|^{b+1}\}$ that have also a 1-bit in the b-th column. Hence, since in the case $s_b = 1$ the computation in Line 7 updates the shares in $x_{n,b}$ so that it equals the product between $x_{n,b+1}$ and $x_{n,b}$, the aposteriori values of $x_{n,b}$ identify the set $\arg\max_n\{x_n|^b\}$. That completes the proof.

□

Protocol 7: A bit-decomposition-based protocol for solving `MMP`

```
Parameters: $N$ - number of inputs; $B > 0$ - the number of bits in each of the
            inputs.
Private Inputs: $x_1, \dots, x_N \in [0, 2^B)$; the bit decomposition of $x_n$ is
                $(x_{n,B-1}, \dots, x_{n,0})$, $n \in [N]$.
1 forall $n \in [N]$ do
2     forall $b = B-1, \dots, 0$ do
3         $[x_{n,b}] \leftarrow (\mathsf{PrivateInput}, n, x_{n,b})$.
4 forall $b = B-1, \dots, 0$ do
5     Compute $[s_b] \leftarrow \bigvee_{n \in [N]}([x_{n,b+1}] \cdot [x_{n,b}])$ .
6     forall $n \in [N]$ do
7         $[x_{n,b}] \leftarrow (1 - [s_b]) \cdot [x_{n,b+1}] + [s_b] \cdot [x_{n,b} \cdot x_{n,b+1}]$.
8 The parties call $(\mathsf{PublicOutput}, [x_{n,0}])$ for all $n \in [N]$.
Output: The indices $n \in [N]$ for which $x_n = M := \max\{x_i : i \in [N]\}$.
```

We proceed to discuss the protocol's complexity. Each party breaks its input to B binary bits and inputs each of them separately, so that there are in total BN calls to PrivateInput (Lines 1-3). Then, there are BN calls to Multiply followed by B calls to Or (Line 5) and $2BN$ additional calls to Multiply (Line 7). Finally, there are N calls to PublicOutput (Line 8). In summary,

$$\mathsf{size}(\text{Protocol 7}) = BN \cdot [\textsc{PrI}] + 3BN \cdot [\textsc{Mul}] + B \cdot [\textsc{Or}]_N + N \cdot [\textsc{PuO}] \,.$$

It implies that

$$\mathsf{preDepth}(\text{Protocol 7}) = \max([\textsc{PrI}], [\textsc{Mul}], [\textsc{Or}]_N, [\textsc{PuO}]) \,.$$

As for the online depth, let us concentrate on one of the B iterations. All multiplications in Line 5 can be executed in parallel. Once they are completed, the Or in Line 5 can be computed. Finally, all $2N$ multiplications in Line 7 can be also parallelized. In summary, we conclude that

$$\mathsf{onDepth}(\text{Protocol 7}) = [\textsc{PrI}] + B \cdot (2[\textsc{Mul}] + [\textsc{Or}]_N) + [\textsc{PuO}] \,.$$

Example. We conclude our discussion of Protocol 7 by exemplifying its operation. Assume that $N = 4$ and that $x_1 = 11$, $x_2 = 7$, $x_3 = 10$, and $x_4 = 11$. In this case $B = 4$ and the inputs' bits are as follows:

	$b=3$	$b=2$	$b=1$	$b=0$
$n=1$	1	0	1	1
$n=2$	0	1	1	1
$n=3$	1	0	1	0
$n=4$	1	0	1	1

.

We begin with the MSB column, $b = 3$. Here, $s_3 = (1\cdot1)\vee(1\cdot0)\vee(1\cdot1)\vee(1\cdot1) = 1$ (since $x_{n,b+1} = x_{n,B} = 1$ in all rows). Hence, the update of bits in that column is $x_{n,3} \leftarrow x_{n,3} \cdot x_{n,4}$ and, because $x_{n,4} = 1$ for all n, the column $b = 3$ remains unchanged.

	$\downarrow$ $b=3$	$b=2$	$b=1$	$b=0$
$n=1$	**1**	0	1	1
$n=2$	**0**	1	1	1
$n=3$	**1**	0	1	0
$n=4$	**1**	0	1	1

.

Next, we deal with $b = 2$. Here $s_2 = (1\cdot0)\vee(0\cdot1)\vee(1\cdot0)\vee(1\cdot0) = 0$. Hence, the update rule is $x_{n,2} \leftarrow x_{n,3}$. Therefore, when completing this iteration we end up with the following table of bits:

	$b=3$	$\downarrow$ $b=2$	$b=1$	$b=0$
$n=1$	1	**1**	1	1
$n=2$	0	**0**	1	1
$n=3$	1	**1**	1	0
$n=4$	1	**1**	1	1

.

Moving to $b = 1$ we see that $s_1 = 1$ and, therefore, the update rule is $x_{n,1} \leftarrow x_{n,1} \cdot x_{n,2}$. As a result, we end up with the following table:

	$b = 3$	$b = 2$	$\overset{\downarrow}{b = 1}$	$b = 0$
$n = 1$	1	1	**1**	1
$n = 2$	0	0	**0**	1
$n = 3$	1	1	**1**	0
$n = 4$	1	1	**1**	1

.

Finally, we reach the LSB, $b = 0$. Here too $s_0 = 1$. Hence, after the update $x_{n,0} \leftarrow x_{n,0} \cdot x_{n,1}$ we get

	$b = 3$	$b = 2$	$b = 1$	$\overset{\downarrow}{b = 0}$
$n = 1$	1	1	1	**1**
$n = 2$	0	0	0	**0**
$n = 3$	1	1	1	**0**
$n = 4$	1	1	1	**1**

.

Indeed, the bits in the column $b = 0$ identify the maximal inputs in rows $n = 1$ and $n = 4$.

5.2 Reducing depth by using larger bases

The number of iterations in the main loop in Protocol 7 directly determines its depth, since each iteration is computed by a constant-depth sub-protocol. The number of iterations equals the number B of bits in the representation of the inputs. Protocol 7 can be modified so that a larger base 2^d, $d > 1$, is used. Such a modification yields a smaller number of iterations – B/d instead of B (for the sake of simplicity we assume that $d|B$). The generalization is described in Protocol 8, which uses a constant-depth sub-protocol for computing maxima (the same technique as used in Protocol 5).

In Protocol 8 the parties decompose their inputs to digits in a 2^d-base. Thus, if $x_n \in [0, 2^B)$, the number of 2^d-digits in it is $D = B/d$, where the digit in the ℓ-th position is denoted $x_{n,\ell}$, $0 \le \ell < D$. Each such digit is an integer in the range $[0, Q-1]$ where $Q = 2^d$. Then, each digit is provided as an input to the protocol by its monotone representation; i.e., the digit $x_{n,\ell}$ is represented by $Q - 1$ bits – $y_{n,\ell,1}, \dots, y_{n,\ell,Q-1}$ – such that $y_{n,\ell,i} = 1$ iff $x_{n,\ell} \ge i$. Therefore, the representation is monotonically decreasing: $y_{n,\ell,i} \ge y_{n,\ell,i+1}$ for every $1 \le i < Q - 1$. The input of all $Q - 1$ bits in the monotone representation of all D digits in all N inputs is done in Lines 1-4.

The variables $r_{n,\ell}$, $n \in [N]$, $\ell = D, \dots, 0$, are binary flags that equal 1 iff the n-th input is still a candidate to be the maximal input after scanning all inputs' digits from the $(D-1)$-th digit down to the ℓ-th digit. Since initially, before the scan starts, all inputs are candidates, those variables are initialized to $r_{n,D} = 1$ for all $n \in [N]$ (Line 5).

Protocol 8: A digit-decomposition-based protocol for solving MMP

```
Parameters: $N$ - number of inputs in $[0, 2^B)$; $d \in \mathbb{N}$ s.t. $d|B$; $Q := 2^d$;
            $D := B/d$ is the number of base-$2^d$ digits in each of the inputs.
Private Inputs: $x_1, \dots, x_N \in [0, 2^B)$; the digit decomposition of $x_n$, $n \in [N]$,
                is $(x_{n,D-1}, \dots, x_{n,0})$, where $x_{n,\ell} \in [0, Q)$, $0 \le \ell \le D-1$
                (that is, $x_n = \sum_{\ell=0}^{D-1} x_{n,\ell} \cdot Q^\ell$).
1  forall $n \in [N]$ do
2      forall $0 \le \ell \le D-1$ do
3          forall $i \in [Q-1]$ do
4              $[y_{n,\ell,i}] \leftarrow (\mathsf{PrivateInput}, n, y_{n,\ell,i})$, where $y_{n,\ell,i} = 1_{x_{n,\ell} \ge i}$.
5  Initialize $[r_{n,D}] \leftarrow 1$ for all $n \in [N]$.
6  forall $\ell = D-1, \dots, 0$ do
7      forall $n \in [N]$ and $i \in [Q-1]$ do
8          $[y'_{n,\ell,i}] \leftarrow [y_{n,\ell,i}] \cdot [r_{n,\ell+1}]$
9      $([M_{\ell,Q-1}], \dots, [M_{\ell,1}]) \leftarrow \mathtt{MaxMonotone}(N, Q, \{[y'_{n,\ell,i}]\}_{n \in [N], i \in [Q-1]})$.
10     forall $n \in [N]$ do
11         $[1_{M_\ell = x_{n,\ell}}] = \mathtt{EqualBits}(Q, \{[y'_{n,\ell,i}]\}_{i \in [Q-1]}, \{M_{\ell,i}\}_{i \in [Q-1]})$.
12         $[r_{n,\ell}] \leftarrow [r_{n,\ell+1}] \cdot [1_{M_\ell = x_{n,\ell}}]$.
13 The parties call $(\mathsf{PublicOutput}, [r_{n,0}])$ for all $n \in [N]$.
Output: The indices $n \in [N]$ for which $x_n = M := \max\{x_i : i \in [N]\}$.
```

The main loop (Lines 6-12) scans the D digits of all inputs, from the most significant one ($\ell = D-1$) to the least significant one ($\ell = 0$). First, the parties multiply the monotone bit representation of the current bit in each of the inputs with the bit that indicates whether that input is still a potential candidate to be the maximum (Lines 7-8). Then, the monotone representations of the current digit in each of the relevant inputs are used to compute the monotone representation of the maximal digit in that position using the sub-protocol `MaxMonotone` (Line 9). Then, we check the digits of all inputs against the computed maximum (Lines 10-11) by calling the sub-protocol `EqualBits`. Subsequently (Line 12), we update $r_{n,\ell}$ to be 1 iff that input is still a candidate at this point ($r_{n,\ell+1} = 1$) and the current digit is maximal ($M_\ell = x_{n,\ell}$). At the conclusion of this loop, the protocol outputs the bits $r_{n,0}$, $n \in [N]$, that identify the maximal inputs (Line 13). The proof of this protocol's correctness is very similar to that of Protocol 7, see Lemma 1.

We now turn to describe the sub-protocols that Protocol 8 invokes. Sub-protocol 9 computes the monotone representation of the maximum of inputs that are also given through their monotone representation. It is called by Protocol 8 in Line 9. Sub-protocol 10, namely $\mathtt{EqualBits}(Q, \{[x_i]\}_{i \in [Q-1]}, \{[x'_i]\}_{i \in [Q-1]})$, is given handles to two bit-vectors and it outputs 1 iff they are equal in every position. We rely here on the straightforward identity $\bigwedge_{i \in [Q-1]} (1_{x_i = x'_i}) = 1 - \left(\bigvee_{i \in [Q-1]} (x_i - x'_i)^2\right)$. This sub-protocol is called in Line 11 of Protocol 8.

Sub-protocol 9: `MaxMonotone`$(N, Q, \{[y_{n,i}]\}_{n\in[N],i\in[Q-1]})$

```
    Parameters: $N$ - number of inputs; $Q$ - an upper bound on the inputs (each
                input is in $[0, Q-1)$.
    Inputs: Handles $[y_{n,i}]$, $n \in [N]$, $i \in [Q-1]$, such that $y_{n,i} \in \{0,1\}$ and
            $y_{n,i} \geq y_{n,i+1}$ for all $i \in [Q-2]$ and $n \in [N]$.
1   forall $i \in [Q-1]$ do
2       $[M_i] \leftarrow \bigvee_{n\in[N]}[y_{n,i}]$.
    Output: The handles $([M_{Q-1}], \ldots, [M_1])$ to the monotone representation of
            $M := \max_{n\in[N]}\{\max_{i\in[Q-1]}\{y_{n,i} = 1\}\}$.
```

Sub-protocol 10: `EqualBits`$(Q, \{[x_i]\}_{i\in[Q-1]}, \{[x'_i]\}_{i\in[Q-1]})$

```
    Parameters: $Q$ - determines the length of the two bit vectors.
    Inputs: Handles to two bit vectors, $\{[x_i]\}_{i\in[Q-1]}$ and $\{[x'_i]\}_{i\in[Q-1]}$.
1   $[1_{x=x'}] \leftarrow 1 - \left(\bigvee_{i\in[Q-1]]}([x_i] - [x'_i])^2\right)$ .
    Output: The handle $[1_{x=x'}]$.
```

The complexity of Protocol 9 is analyzed as follows. First, we analyse the complexity of the sub-protocols. The costs of `MaxMonotone` are:

$$\mathsf{size}(\text{Protocol } 9) = (Q-1)[\textsc{Or}]_N\,,$$

$$\mathsf{preDepth}(\text{Protocol } 9) = [\textsc{Or}]_N\,, \quad \mathsf{onDepth}(\text{Protocol } 9) = [\textsc{Or}]_N\,.$$

The costs of `EqualBits` are:

$$\mathsf{size}(\text{Protocol } 10) = (Q-1)[\textsc{Mul}] + [\textsc{Or}]_{Q-1}\,,$$

$$\mathsf{preDepth}(\text{Protocol } 10) = \max([\textsc{Mul}], [\textsc{Or}]_{Q-1})\,, \quad \mathsf{onDepth}(\text{Protocol } 10) = [\textsc{Mul}]+[\textsc{Or}]_{Q-1}\,.$$

Hence, Protocol 8's costs are:

$$\begin{aligned}\mathsf{size}(\text{Protocol } 8) &= ND(Q-1)[\textsc{PrI}] + ND(Q-1)[\textsc{Mul}] + D[\texttt{MaxMonotone}]\\ &\quad + ND\big([\texttt{EqualBits}] + [\textsc{Mul}]\big) + N[\textsc{PuO}]\\ &= ND(Q-1)[\textsc{PrI}] + ND(Q-1)[\textsc{Mul}] + D(Q-1)[\textsc{Or}]_N\\ &\quad + ND\big((Q-1)[\textsc{Mul}] + [\textsc{Or}]_{Q-1} + [\textsc{Mul}]\big) + N[\textsc{PuO}]\\ &= ND(Q-1)[\textsc{PrI}] + ND(2Q-1)[\textsc{Mul}] + ND[\textsc{Or}]_{Q-1}\\ &\quad + D(Q-1)[\textsc{Or}]_N + N[\textsc{PuO}]\,,\end{aligned}$$

$$\mathsf{preDepth}(\text{Protocol } 8) = \max\left([\textsc{PrI}], [\textsc{Mul}], [\textsc{Or}]_N, [\textsc{Or}]_{Q-1}, [\textsc{PuO}]\right),$$

$$\begin{aligned}\mathsf{onDepth}(\text{Protocol } 8) &= [\textsc{PrI}] + D\big([\textsc{Mul}] + [\texttt{MaxMonotone}] + [\texttt{EqualBits}] + [\textsc{Mul}]\big)\\ &\quad + [\textsc{PuO}] = [\textsc{PrI}] + D\big(3[\textsc{Mul}] + [\textsc{Or}]_N + [\textsc{Or}]_{Q-1}\big) + [\textsc{PuO}]\,.\end{aligned}$$

6 A protocol for solving MMP inspired by [1]

The study of Aggarwal et al. [1] presented a secure protocol for finding the k-th ranked element in the union of private datasets. It can be used to solve MaxP, and subsequently MMP, if we view the input to those problems as N private singleton datasets and take $k = N$. We proceed to describe such a protocol, for the sake of comparing it with the previous protocols that were presented in this study.

Protocol 11 computes the maximum by implementing a privacy-preserving binary search over the interval $[0, 2^B - 1]$, where $2^B - 1$ is an upper bound on all inputs (Lines 1-15). The lower and upper bounds of the search interval, denoted a and b, are initialized in Line 1. A binary flag f is set to zero in Line 2; it will be set to 1 once the maximum is found. The binary search loop is given in Lines 3-15. The contemplated value of the maximum, M, is set to the middle of the current search interval (Line 4). Then, each of the parties set two local flags, ℓ_n and g_n, that indicate whether its input is larger or smaller than M (Lines 5-7). Subsequently, all determine jointly whether there is at least one input larger than M ($u = 1$) or not ($u = 0$), and whether all inputs are smaller than M ($v = 1$) or not ($v = 0$) (Line 8). This is done by computing the OR (AND) of the private bits ℓ_n (g_n).

If $x_n \leq M$ for all n ($u = 0$) and there exists at least one input x_j such that $x_j \geq M$ ($v = 0$) then M is the sought-after maximum. In that case f is set to 1 so that the loop terminates (Lines 9-10). Otherwise, if $u = 1$ then there is at least one input larger than M so that the sought maximum is at least $M + 1$. In that case we update the search lower bound to that value (Line 11-12). Otherwise, if $v = 1$ then all inputs are smaller than M and hence the search upper bound is updated to $M - 1$ (Lines 13-14).

In the concluding stage, the protocol finds all private inputs that equal the maximum (Lines 16-17) and outputs them.

7 Concrete evaluation

The goal of this section is to arm the reader with a methodology of measuring and comparing the protocols presented in this paper for a given implementation of the ABB functionality in a given setting. To demonstrate the methodology, we pick the statistically secure implementation of Damgård and Nielsen [11], in the setting of semi-honest adversary, who statically corrupts a minority of the parties (also known as the honest majority setting). In Section 7.1 we show an implementation of the 'high-level' interfaces using the 'low-level' ones; then we formulate the costs of all interfaces based on the specific parameters of the Multiple Millionaires' Problem in Section 7.2; these parameters consist of the number of parties, number of inputs and the bit-length of the inputs; finally, in Section 7.3 we compare the performance of the protocol for a set of specific parameters.

Protocol 11: Solving `MMP` by binary search [1]

```
    Private inputs: $P_n$ has $x_n \in [0, 2^B - 1]$, $n \in [N]$.
 1  Set $a \leftarrow 0$ and $b \leftarrow 2^{B-1}$
 2  Set $f \leftarrow 0$.
 3  repeat
 4      Set $M \leftarrow \lceil (a+b)/2 \rceil$
 5      forall $n \in [N]$ do
 6          $P_n$ sets $\ell_n \leftarrow 1$ if $x_n > M$ and $\ell_n \leftarrow 0$ otherwise.
 7          $P_n$ sets $g_n \leftarrow 1$ if $x_n < M$ and $g_n \leftarrow 0$ otherwise.
 8      The parties set $u \leftarrow \bigvee_n \ell_n$ and $v \leftarrow \bigwedge_n g_n$.
 9      if $u = 0$ and $v = 0$ then
10          Set $f \leftarrow 1$.
11      else if $u = 1$ then
12          Set $a \leftarrow M + 1$.
13      else if $v = 1$ then
14          Set $b \leftarrow M - 1$.
15  until $f = 1$.
16  forall $n \in [N]$ do
17      The parties compute $1_{x_n = M}$.
    Output: The indices $n \in [N]$ for which $x_n = M := \max\{x_i : i \in [N]\}$.
```

7.1 Interfaces

Here, we break down the high-level interfaces (like Equality and OR) into their cost in terms of more primitive interfaces (like PrivateInput and Multiply), and then derive their concrete costs, for a specific setting that we selected. Such a detailed analysis is necessary in order to enable a comparison between protocols and in order to get a more precise estimate of the costs of a given protocol. However, we stress that such a breakdown analysis might be different for different settings and, hence, it should be carried out independently for each such setting.

The cost analysis of interfaces [PRI], [PUO], [RND], [MUL] is based on the passively secure MPC protocol of Damgård and Nielsen [11]; the cost analysis of interface [INV] is trivial and given in Appendix A.2.1; and the cost of interfaces [COM] and $[\text{OR}]_m$ is as derived in [17] and [26], respectively.

We examine a setting where the ABB is implemented by a protocol that guarantees statistical security in the presence of a semi-honest adversary. We choose a semi-honest adversary in order to avoid noisy costs that are typically added to ensure input consistency when the adversary is malicious. These costs might distract us from a clean and objective comparison between the protocols.

In our discussion we distinguish between the number of inputs, N, and the number of MPC parties, K. For example, if the MPC parties are the owners of the inputs, where party P_k holds n_k private inputs, then $N = \sum_{k=1}^{K} n_k$. In another possible scenario, the N inputs are held by external clients who distribute shares in them to K servers who act as the MPC parties. In our calculation of the protocol's size we are interested in the communication complexity per MPC party (the number of field elements it sends/receives).

The setting that we picked allows us to use the concrete costs of Or and Equality as given in Appendix A.2. In the following we present the final costs, after integrating the concrete costs from Appendix A.2. Let us first provide concrete costs to some of the interfaces in Functionality 1:

- PrivateInput: Submitting a private input boils down to a simple a secret sharing, in which the party who acts as the dealer sends 1 field element to every other party, in 1 round. Hence,

$$\mathsf{size}([\textsc{PrI}]) = 1\,, \quad \mathsf{preDepth}([\textsc{PrI}]) = 0\,, \quad \mathsf{onDepth}([\textsc{PrI}]) = 1.$$

- PublicOutput is implemented by opening the secret to a single designated party, who then announces the result to all other parties. This translates into a communication of 2 field elements per party, in 2 rounds. The number of communication rounds may be reduced, at the expense of increasing the communication cost, in the following manner: each party sends its share in the output to all other parties; that entails sending K field elements in 1 round. We use the latter protocol as the basis to our calculations, hence

$$\mathsf{size}([\textsc{PuO}]) = K\,, \quad \mathsf{preDepth}([\textsc{PuO}]) = 0\,, \quad \mathsf{onDepth}([\textsc{PuO}]) = 1.$$

- Random: Generating a single secret random field element and distributing shares in it entails communicating 1 field element (per party) in 1 round. Hence,

$$\mathsf{size}([\textsc{Rnd}]) = 1\,, \quad \mathsf{preDepth}([\textsc{Rnd}]) = 1, \quad \mathsf{onDepth}([\textsc{Rnd}]) = 0.$$

- (Multiply, $[a], [b]$) is implemented by the DoubleRandom technique presented in [11]. The DoubleRandom procedure outputs two handles, $[r]$ and $[R]$, to the same random field element, where $[r]$ is a sharing of degree t and $[R]$ is a sharing of degree $2t$, and t is an upper bound on the number of corrupted parties. The parties locally compute $[C] = [a] \cdot [b]$, in order to get a sharing $[C]$ of the product $a \cdot b$ which is of degree $2t$. Then they perform PublicOutput on $c' = [C] - [R]$ and locally compute $[c] = [a \cdot b] = c' + [r]$. DoubleRandom requires twice more communication than Random and the same round complexity (which is one round). Overall, the communication cost is $2 + K$ field elements per party, and 2 rounds, where the DoubleRandom's round occurs in the preprocessing stage, whereas the PublicOutput's round occurs in the online phase. Then,

$$\mathsf{size}([\textsc{Mul}]) = 2 + K\,, \quad \mathsf{preDepth}([\textsc{Mul}]) = 1\,, \quad \mathsf{onDepth}([\textsc{Mul}]) = 1.$$

- (Inverse, $[a]$) can be computed by invoking Random, Multiplly and PublicOutput (see Appendix A.2.1). The costs of that computation are as follows:

$$\mathsf{size}([\textsc{Inv}]) = 3 + 2K\,, \quad \mathsf{preDepth}([\textsc{Inv}]) = 1, \quad \mathsf{onDepth}([\textsc{Inv}]) = 2.$$

– ($\mathsf{Equality}, [a], b$) too (like $\mathsf{Inverse}$) can be computed by invoking Random, $\mathsf{Multiplly}$ and $\mathsf{PublicOutput}$ (see Appendix A.2.2). Hence, its costs are:

$$\mathsf{size}([\textsc{Equ}]) = 3 + 2K\,, \quad \mathsf{preDepth}([\textsc{Equ}]) = 1, \quad \mathsf{onDepth}([\textsc{Equ}]) = 2.$$

– ($\mathsf{Compare}, [a], [b]$): We use the comparison protocol of Goss and Jiang [17]. The costs of that protocol are as follows:

$$\mathsf{size}([\textsc{Com}]) = 2B{+}5\log B{+}23\,, \quad \mathsf{preDepth}([\textsc{Com}]) = 0\,, \quad \mathsf{onDepth}([\textsc{Com}]) = 4\,.$$

– ($\mathsf{OR}_m, [a_1], \ldots, [a_m]$): Nishide and Ohta [26] designed a protocol for computing $\left[\bigvee_{i\in[m]} a_i\right]$ in constant depth. We defer the description of their solution to Appendix A.2.3. The costs of their protocol are:

$$\begin{aligned}\mathsf{size}([\textsc{Or}]_m) &= 2m \cdot [\textsc{Rnd}] + (3m-1)\cdot[\textsc{Mul}] + 2m\cdot[\textsc{PuO}]\\ &= 2m + (3m-1)(2+K) + 2mK = 5NK + 8m - K - 2\,.\end{aligned}$$

In the preprocessing stage there are calls to Random, followed by $\mathsf{Multiply}$ and then $\mathsf{PublicOutput}$, which are dependent on each other. Therefore, the preprocessing depth equals the sum of the *total* depths of these interfaces, which is 4, because $\mathsf{depth}(\mathsf{Random}) = \mathsf{depth}(\mathsf{PublicOutput}) = 1$ and $\mathsf{depth}(\mathsf{Multiply}) = 2$. However, note that $\mathsf{Multiply}$ consists of an invocation of Random on its own, which can be executed in the first round (rather than in the second), and that saves one round. Overall, there are 3 rounds in the preprocessing stage:

$$\mathsf{preDepth}([\textsc{Or}]_m) = [\textsc{Rnd}] + [\textsc{Mul}] + [\textsc{PuO}] = 3\,.$$

As for the online depth, $\mathsf{onDepth}([\textsc{Or}]_m) = 3\cdot[\textsc{Mul}] + [\textsc{PuO}] = 4$.

7.2 Protocols' costs

Now we may proceed to derive the costs of the protocols. We do so by taking the costs (size and depth) of the protocols as we analyzed them in Sections 3–6 and plugging in them the costs of the interfaces that they invoke, as given in Section 7.1. The parameters that determine those costs are: N - number of inputs, K - number of MPC parties, and B - number of bits for representing all inputs. We defer the detailed computation of those costs to Appendix B. Note that those costs were already presented in Table 1 (where the size is shown there only asymptotically, while the preprocessing and online depths are shown exactly).

7.3 Concrete costs in specific scenarios

We present the concrete cost for two scenarios: in the first (Figure 1, top), the inputs x_n are drawn from a small domain, $0 \le x_n < 256$ and so $B = 8$; and in the second (Figure 1, bottom), the inputs are drawn from a large domain, $0 \le x_n < 2^{32}$, and so $B = 32$. In the figure, the protocols are distinguished by color and line style: Protocol 1 is presented by a solid black line; Protocol 4 has three versions, each with a different setting of N', presented by green lines (solid, dashed and dotted); Protocol 6 is presented by a solid blue line; Protocol 7 is presented by a pink line; Protocol 8 is presented by orange lines, where the line style (solid, dashed or dotted) identify the setting of d, the digit length; and Protocol 11 is presented by a solid cyan line. The two plots in the figure present the trend of the costs as the number of inputs, N, grows. We evaluate the costs for $N \in \{2^8, 2^{16}, 2^{24}, 2^{32}\}$, and each setting is presented by a different marker (see the legend on the left side of each of the two plots). For other scenarios, the reader may access the notebook with the source code that we used to calculate these costs and change the parameters accordingly, see https://colab.research.google.com/drive/1yjEt0lhFteSIUb06ipjxCzKSnnE5PTfA?usp=drive_link.

When latency is at premium (e.g., when a system invokes MMP repetitively) one would prefer using a low depth protocol. The comparison in Figure 1 suggests that when the inputs are taken from a small domain (e.g., $B = 8$) then the monotone-representation-based protocol (Protocol 6) performs best. If, however, the inputs are taken from larger domains (e.g., $B = 32$), Protocol 6 becomes impractical and then one should consider Protocol 8 or Protocol 4, with a suitable choice of their parameters (d and N', respectively). On the other hand, when bandwidth is at premium, the "naïve" binary-tree-based protocol (Protocol 1) performs better than others.

8 Related work

While many papers considered the (two party) Millionaires' Problem, its multi-party generalization, MMP, is introduced and studied herein for the first time (to the best of our knowledge). We review here some of the papers that addressed problems similar to MMP and MaxP.

Liu et. al. [23] considered a problem that is related to MaxP. They assumed that all inputs in $I := \{x_n : n \in [N]\}$ are distinct non-negative integers that are bounded by a small integer Q. They devised a protocol that outputs the set I completely, while perfectly hiding the mapping between I and the set of parties $\{P_n : n \in [N]\}$. While such a solution reveals the output of MaxP, it reveals far more than what is desired in MaxP and MMP. In addition, the complexity of their protocol scales linearly with Q (which is equivalent to our 2^B), what limits significantly its scalability.

The studies [28,29] concentrate on the sequencing problem: the setting in that problem is as in MMP and MaxP but the required output is a set of ranks $\{r_m : m \in [N]\}$, where $r_m := |\{n \in [N] : x_n > x_m\}|$, $m \in [N]$. Hence, all inputs x_m, $m \in [N]$, for which $r_m = 0$ are the maximum. Here, as opposed to the above mentioned study of Liu et. al. [23], the values of the inputs remain hidden. But the output is still far more elaborated than what is desired in MMP and MaxP.

Aggarwal et al. [1] considered a setting with two or more parties possessing large confidential datasets and designed secure protocols for computing the k-th ranked element of the union of the datasets. Their protocol template FIND-RANKED-ELEMENT-MULTIPARTY can be translated into a protocol for solving

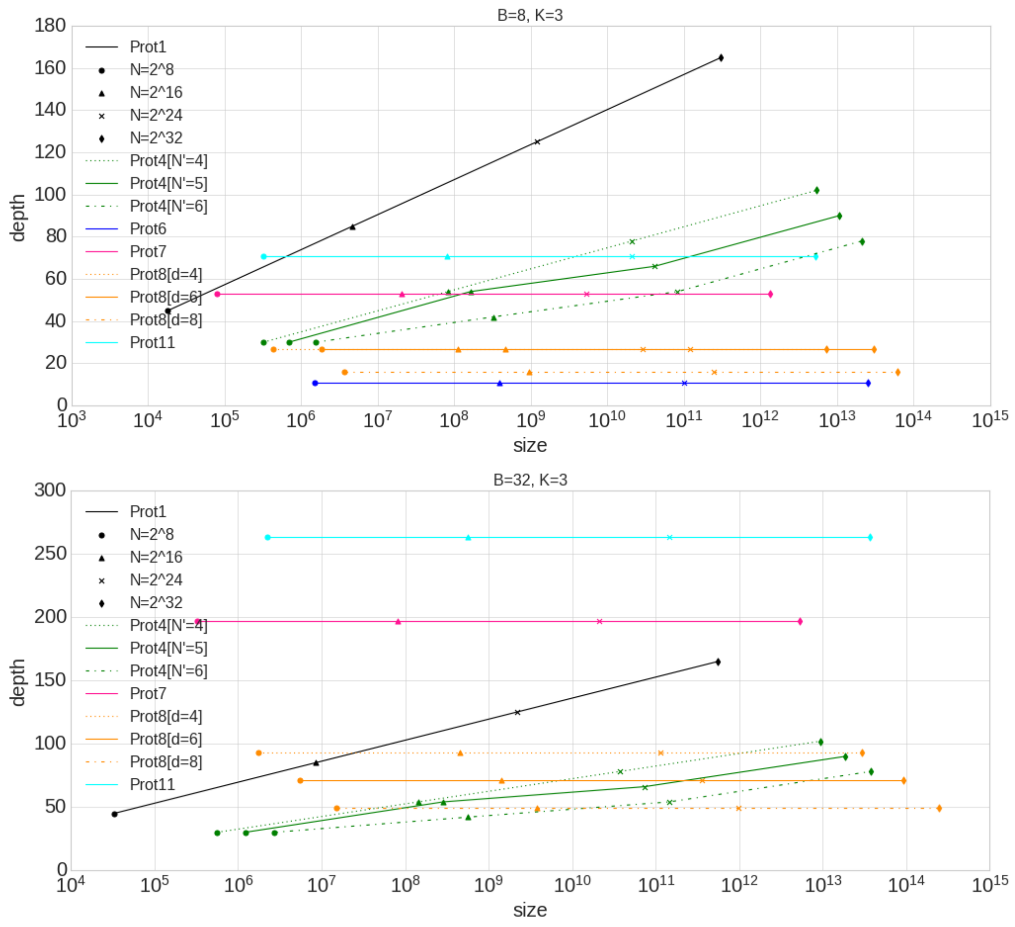

Fig. 1. The size and depth the protocols presented in this paper.

MaxP. We presented that protocol as Protocol 11 in Section 6. The solution to MaxP as issued by Protocol 11 can be securely converted to a solution of MMP (as we do in Lines 12-13 in Protocol 1). In Section 6 we also analyzed the protocol's size and depth, and included it in our comparison in Table 1 in the Introduction and in Figure 1 in Section 7.3.

David et al. [12] and Mohassel et. al. [24] solved the MMP problem as a building block for constructing privacy-preserving machine learning protocols. The approach taken by these works is similar to the binary-tree-based protocol (Protocol 1) and has the same costs. Namely, they solve MMP by reducing it to a sequence of (two party) MPs. In [12], they also mentioned a 'flattened tree' approach, in which the tree's height is 1. This means that each leaf is compared to every other leaf at the same round. This approach is captured by our general-tree-based protocol (Protocol 4) by setting $N' = N$.

Recall that our bit-decomposition and digit-decomposition based protocols (Protocols 7 and 8, respectively) have depth *linear* in the number of digits, D ($1 \leq D \leq B$). On the other hand, protocols that solve the original (two-party) Millionaires' problem have depth that is only *logarithmic* in D, see [9,14,27]. These protocols use the following recurrence relation: to compare two numbers x and y of B bits each, we can split them to the four integers $[x_{\mathsf{high}}, x_{\mathsf{low}}]$ and $[y_{\mathsf{high}}, y_{\mathsf{low}}]$, where x_{high} (resp. y_{high}) is of B_{high} bits, x_{low} (resp. y_{low}) is of $B_{\mathsf{low}} := B - B_{\mathsf{high}}$ bits, and $x = x_{\mathsf{high}} \cdot 2^{B_{\mathsf{low}}} + x_{\mathsf{low}}$ (resp. $y = y_{\mathsf{high}} \cdot 2^{B_{\mathsf{low}}} + y_{\mathsf{low}}$). This splitting enables computing the comparison bit $1_{x<y}$ through the equation

$$1_{x<y} = 1_{x_{\mathsf{high}}<y_{\mathsf{high}}} \vee \left(1_{x_{\mathsf{high}}=y_{\mathsf{high}}} \wedge 1_{x_{\mathsf{low}}<y_{\mathsf{low}}}\right), \tag{1}$$

which leads to a binary tree of comparisons of depth $\log D$. Let us try to extend the above recurrence relation to MMP, and let us consider the case $N = 3$. Letting x, y, z be the three private inputs, each consisting of B bits, and each is split into lower and higher parts, as described above, then the natural extension of Eq. (1) would be as follows:

$$1_{x=\max(x,y,z)} = 1_{x_{\mathsf{high}}=\max(x_{\mathsf{high}},y_{\mathsf{high}},z_{\mathsf{high}})} \vee \left(1_{x_{\mathsf{high}}=y_{\mathsf{high}}=z_{\mathsf{high}}} \wedge 1_{x_{\mathsf{low}}=\max(x_{\mathsf{low}},y_{\mathsf{low}},z_{\mathsf{low}})}\right).$$

However, it is incorrect. For instance, if $x_{\mathsf{high}} = y_{\mathsf{high}} > z_{\mathsf{high}}$ and $y_{\mathsf{low}} > x_{\mathsf{low}}$, then a protocol based on that relation would output both x and y as the maximal values, whereas the correct output is only y. Therefore, it is not clear whether the two-party recurrence relation can be extended to the multi-party setting.

9 Conclusion

We studied here two fundamental MPC problems — `MMP` and `MaxP`. Those problems are natural extensions of Yao's classical Millionaires' Problem [31]. As applications of privacy-preserving computation are more and more commonly implemented in industrial systems, `MMP` and `MaxP` become important building blocks in privacy-preserving statistics, machine learning, auctions and other domains.

While some prior studies considered problems related to `MMP` [1,23,28,29], and others did solve `MMP` by reducing it to a sequence of `MP`s [12,24], it appears that ours is the first study that introduces this fundamental MPC problem and offers dedicated solutions, of different approaches, and systematically compares between them in order to illustrate the tradeoff between the size and the depth of the corresponding protocols.

A prominent advantage of our novel protocols is their simplicity. As they solve fundamental problems that are essential building blocks in important applications, our systematic study of solutions to those basic problems, and the comparison between them, will serve well future researchers of MPC and practitioners of secure distributed computing.

References

1. Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the median (and other elements of specified ranks). *J. Cryptol.*, 23:373–401, 2010.
2. Ramiro Alvarez and Mehrdad Nojoumian. Comprehensive survey on privacy-preserving protocols for sealed-bid auctions. *Comput. Secur.*, 88, 2020.
3. Ian F. Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *ASIACRYPT*, pages 515–529, 2004.
4. Marina Blanton and Paolo Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*, pages 190–209, 2011.
5. Paul Bunn and Rafail Ostrovsky. Secure two-party k-means clustering. *IACR Cryptol. ePrint Arch.*, page 231, 2007.
6. Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):143–202, 2000.
7. Yao-Jen Chang, Chia-Wei Tsai, and Tzonelih Hwang. Multi-user private comparison protocol using GHZ class states. *Quantum Inf. Process.*, 12:1077–1088, 2013.
8. Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya P. Razenshteyn, and M. Sadegh Riazi. SANNS: scaling up secure approximate k-nearest neighbors search. In *USENIX*, pages 2111–2128, 2020.
9. Geoffroy Couteau. New protocols for secure equality test and comparison. In *ACNS*, pages 303–320, 2018.
10. Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *CRYPTO*, pages 247–264, 2003.
11. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.
12. Bernardo Machado David, Rafael Dowsley, Raj S. Katti, and Anderson C. A. Nascimento. Efficient unconditionally secure comparison and privacy preserving machine learning classification protocols. In *ProvSec*, pages 354–367, 2015.
13. Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Topics in Cryptology - CT-RSA*, pages 457–472, 2001.
14. Juan A. Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *PKC*, pages 330–342, 2007.
15. Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In David B. Shmoys, editor, *STOC 2014*, pages 495–504. ACM, 2014.
16. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
17. Ken Goss and Wei Jiang. Efficient and constant-rounds secure comparison through dynamic groups and asymmetric computations. *IACR Cryptol. ePrint Arch.*, page 179, 2018.
18. Dima Grigoriev, Laszlo B. Kish, and Vladimir Shpilrain. Yao's millionaires' problem and public-key encryption without computational assumptions. *Int. J. Found. Comput. Sci.*, 28:379–390, 2017.
19. Yan Huang, Lior Malka, David Evans, and Jonathan Katz. Efficient privacy-preserving biometric identification. In *NDSS*, 2011.
20. Ioannis Ioannidis and Ananth Grama. An efficient protocol for yao's millionaires' problem. In *Hawaii International Conference on System Sciences (HICSS)*, page 205, 2003.

21. Kimmo Järvinen, Helena Leppäkoski, Elena Simona Lohan, Philipp Richter, Thomas Schneider, Oleksandr Tkachenko, and Zheng Yang. PILOT: practical privacy-preserving indoor localization using outsourcing. In *European Symposium on Security and Privacy, EuroS&P*, pages 448–463, 2019.
22. Hsiao-Ying Lin and Wen-Guey Tzeng. An efficient solution to the millionaires' problem based on homomorphic encryption. In *ACNS*, pages 456–466, 2005.
23. Xin Liu, Shundong Li, Xiubo Chen, Gang Xu, Xiaolin Zhang, and Yong Zhou. Efficient solutions to two-party and multiparty millionaires' problem. *Secur. Commun. Networks*, 2017:5207386:1–5207386:11, 2017.
24. Payman Mohassel, Mike Rosulek, and Ni Trieu. Practical privacy-preserving k-means clustering. *Proc. Priv. Enhancing Technol.*, 2020:414–433, 2020.
25. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *EC-99*, pages 129–139, 1999.
26. Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *PKC*, pages 343–360, 2007.
27. Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *CCS*, pages 325–342, 2020.
28. Yi Sun, Qiaoyan Wen, Yudong Zhang, Hua Zhang, and Zhengping Jin. Efficient secure multiparty computation protocol for sequencing problem over insecure channel. *Artificial Intelligence and Its Applications*, 2013. Article ID 172718.
29. ChunMing Tang, GuiHua Shi, and ZhengAn Yao. Secure multi-party computation protocol for sequencing problem. *Sci. China Inf. Sci.*, 54:1654–1662, 2011.
30. Tomas Toft. Constant-rounds, almost-linear bit-decomposition of secret shared values. In *CT-RSA*, pages 357–371, 2009.
31. Andrew C. Yao. Protocols for secure computation. In *FOCS*, pages 160–164, 1982.

A Some comments about the ABB functionality

A.1 Reducing the ABB functionality

We note that some of the interfaces in the ABB functionality 1 could be realized using other ones. Specifically:

- Given the handle $[a]$, we could use the $\mathsf{PrivateInput}$ and $\mathsf{PublicOutput}$ interfaces to realize the $\mathsf{PrivateOutput}$ interface: party P_n (to whom the private output is destined) calls $(\mathsf{PrivateInput}, n, r)$ with some uniformly random value $r \in \mathbb{F}$, followed by the other parties calling $(\mathsf{PrivateInput}, n)$. Then the parties compute $[\Delta_a] \leftarrow [a] - [r]$ and call $(\mathsf{PublicOutput}, [\Delta_a])$, after which P_n computes $a = \Delta_a + r = a - r + r = a$. Clearly, while P_n gets the desired output, all other parties learn nothing in the process.
- Given the handle $[b]$, it is possible to realize $\mathsf{Duplicate}$ as follows: the parties call $[z] \leftarrow (\mathsf{PublicInput}, 0)$ and then they compute $[b] \leftarrow [a] + [z]$.
- Given the handles $[a_1], \ldots, [a_k]$ it is possible to realize $\mathsf{AffineComb}$ as follows: the parties call $[c_i] \leftarrow (\mathsf{PublicInput}, c_i)$ for all $0 \leq i \leq k$ and then they compute $[c] \leftarrow [c_0] + [c_1] \cdot [a_1] + \cdots [c_k] \cdot [a_k]$.
- Given handles $[a]$ and $[b]$, it is possible to realize the $\mathsf{Compare}$ interface using other interfaces in Functionality 1. There are many such reductions, see [17] and the reference therein.
- Given the handle $[a]$, it is possible to realize Equal to some value b by calling $[b] \leftarrow (\mathsf{PublicInput}, b)$, and then computing $[1_{a<b}]$ and $[1_{b<a}]$. Clearly, $a = b$ if and only if $1_{a<b} = 1_{b<a} = 0$. So the parties compute $[1_{a=b}] \leftarrow [1 - 1_{a<b}] \cdot [1 - 1_{b<a}]$. We present a more efficient instantiation in Appendix A.2.2.

A.2 Secure computation of some of the ABB interfaces

A.2.1 Secure computation of an inverse Given a handle $[a]$ to an element $a \in \mathbb{F} \setminus \{0\}$, we want to obtain a handle $[b]$ to $b = a^{-1}$, namely, the inverse of a in the field $\mathbb{F}$. This can be done as follows:

1. The parties invoke $[\textsc{Rnd}]$ and obtain the handle $[c]$ for a random $c \in \mathbb{F}$.
2. The parties compute $[C] = [a \cdot c]$.
3. The parties invoke $[\textsc{PuO}]$ on the handle $[C]$.
4. The parties locally compute $[a^{-1}] \leftarrow [c] \cdot C^{-1}$, and output that handle.

Hence, the costs of this computation are:

$$\mathsf{size}([\textsc{Inv}]) = [\textsc{Rnd}] + [\textsc{Mul}] + [\textsc{PuO}] \,,$$

$$\mathsf{preDepth}([\textsc{Inv}]) = \max([\textsc{Rnd}], [\textsc{Mul}], [\textsc{PuO}]) \,,$$

$$\mathsf{onDepth}([\textsc{Inv}]) = [\textsc{Rnd}] + [\textsc{Mul}] + [\textsc{PuO}] \,.$$

A.2.2 Secure testing of equality The Equality interface can be instantiated as follows: Given $[a]$ and b, compute $[z] = [a] - b$ and test whether the value behind the handle $[z]$, denoted z, equals zero. Equality to zero can be tested by multiplying z with a random value from the field, r, and output the result. If the result is zero then it implies that z is zero (with high probability - except in the case $r = 0$). Otherwise, it implies with certainty that z is non-zero. Obviously, in order to not reveal z (in case it is not zero) we must keep r secret. So the parties invoke $[r] \leftarrow$ Random, then compute $[t] = [r] \cdot [z]$ and then PublicOutput($[t]$). Therefore, the costs of Equality are:

$$\mathsf{size}([\textsc{Equ}]) = [\textsc{Rnd}] + [\textsc{Mul}] + [\textsc{PuO}]\,,$$

$$\mathsf{preDepth}([\textsc{Equ}]) = \max([\textsc{Rnd}], [\textsc{Mul}], [\textsc{PuO}])\,.$$

and

$$\mathsf{onDepth}([\textsc{Equ}]) = [\textsc{Rnd}] + [\textsc{Mul}] + [\textsc{PuO}]\,.$$

A.2.3 Secure computation of the OR operation [26] The idea in [26] was to consider the scalar $A = 1 + \sum_{i\in[m]} a_i \in \{1, \ldots, m+1\}$, and the unique m-degree polynomial $f(x) = \sum_{i=0}^{m} \alpha_i x^i$ that satisfies $f(1) = 0$ and $f(i) = 1$ for all $i \in \{2, \ldots, m+1\}$. Since $f(A) = \bigvee_{i\in[m]} a_i$, the protocol evaluates $f(A) = \sum_{i=0}^{m} \alpha_i A^i$. To do this, the protocol proceeds as follows.

First, the parties perform a sub-protocol that generates m *secret* random values b_i, $i \in [m]$, as well as their inverses, b_i^{-1}.[5] The sub-protocol starts by the parties invoking Random $2m$ times in order to obtain handles to $2m$ secret random values: $[b_i]$ and $[b'_i]$, $i \in [m]$. Then, they compute $[B_i] = [b_i] \cdot [b'_i]$ for every $i \in [m]$ and call PublicOutput($[B_i]$). Finally, they locally compute $[b_i^{-1}] = B^{-1} \cdot [b'_i]$. Hence, the size of this sub-protocol is $2m \cdot [\textsc{Rnd}] + m \cdot [\textsc{Mul}] + m \cdot [\textsc{PuO}]$, and its depth is $[\textsc{Rnd}] + [\textsc{Mul}] + [\textsc{PuO}]$. Note that this depth affects only the preprocessing depth of the protocol.

To compute $f(A)$ the parties proceed as follows. First they compute

$$\begin{aligned}
[c_1] &= [A] \cdot [b_1^{-1}] \\
[c_2] &= [A] \cdot [b_1] \cdot [b_2^{-1}] \\
&\vdots \\
[c_{m-1}] &= [A] \cdot [b_{m-2}] \cdot [b_{m-1}^{-1}] \\
[c_m] &= [A] \cdot [b_{m-1}] \cdot [b_m^{-1}]
\end{aligned}$$

Note that $c_1, \ldots, c_m$ can be computed in parallel in a depth-2 sub-protocol, since each of the values is the result of (up to) two products. Then, they call

[5] Note that the parties could randomly generate $b_i = 0$. However, as the underlying field $\mathbb{F}$ is large, the probability of such event is negligible. In addition, if the parties do generate $b_i = 0$, then they will discover it during the invocation of inverse $\textsc{Inv}$ (see Section A.2.1) and then they could select a new random value.

$\mathsf{PublicOutput}(c_i)$ for all $i \in [m]$. Subsequently, each party computes locally $[A^i] = [b_i] \cdot \prod_{j=1}^{i} c_i$ for all $i = 2, \ldots, m$. Finally they compute $[\bigvee_{i=1}^{m} a_i] = [f(A)] = \sum_{i=0}^{m} \alpha_i \cdot [A^i]$. Hence, the size of above computation is $(2m-1) \cdot [\textsc{Mul}] + m \cdot [\textsc{PuO}]$, and its depth is $2[\textsc{Mul}] + [\textsc{PuO}]$.

Combining all of the above we get:

$$\mathsf{size}([\textsc{Or}]) = 2m \cdot [\textsc{Rnd}] + (3m-1) \cdot [\textsc{Mul}] + 2m \cdot [\textsc{PuO}]\,,$$

$$\mathsf{preDepth}([\textsc{Or}]) = [\textsc{Rnd}] + [\textsc{Mul}] + [\textsc{PuO}]$$

and

$$\mathsf{onDepth}([\textsc{Or}]) = 3 \cdot [\textsc{Mul}] + [\textsc{PuO}]\,.$$

B Derivation of the protocols' costs

Protocol 1. The size and depth of this protocol were analyzed in Section 3.1. They depend on the interfaces $[\textsc{PrI}]$, $[\textsc{Com}]$, $[\textsc{Mul}]$, $[\textsc{Equ}]$, and $[\textsc{PuO}]$. Plugging the concrete costs of those interfaces as given in Section 7.1 we arrive at the following concrete costs of Protocol 1:

$$\begin{aligned}\mathsf{size}(\text{Protocol 1}) &= N + (N-1)(2B + 5\log B + 23) + (N-1)(2+K) + N(3+2K) + NK \\ &= N(2B + 5\log B + 29 + 4K) - (2B + 5\log B + 25 + K)\,,\end{aligned}$$

$$\mathsf{preDepth}(\text{Protocol 1}) = 1\,,$$

$$\mathsf{onDepth}(\text{Protocol 1}) = 1 + \lceil \log N \rceil \cdot (4+1) + 2 + 1 = 5 \cdot \lceil \log N \rceil + 4\,.$$

Protocol 4. This protocol depends on another parameter, N', which is the number of items passed to Sub-protocol 3 (`MaxP`). We begin by analyzing the size and depth of Sub-protocol 3:

$$\begin{aligned}\mathsf{size}(\text{Sub-protocol 3}) &= ({N'}^2 - N')(2B + 5\log B + 23) \\ &\quad + N' \cdot (5N'K + 8N' - K - 2) + (N'+1)(2+K) + 3 + 2K \\ &= ({N'}^2 - N')(2B + 5\log B + 23) + {N'}^2(5K+8) + 5 + 3K\,,\end{aligned}$$

$$\mathsf{preDepth}(\text{Sub-protocol 3}) = 3\,, \quad \mathsf{onDepth}(\text{Sub-protocol 3}) = 12\,.$$

Now we turn to analyze Protocol 4. The number of times that it invokes Sub-protocol 3 (`MaxP`) was denoted in Section 3.2 by $T(N, N')$. It approximately equals $\sum_{\ell=1}^{\lceil \log_{N'} N \rceil} \frac{N}{(N')^\ell} \approx \frac{N-1}{N'-1}$. Hence, the costs of Protocol 4 are:

$$\mathsf{size}(\text{Protocol 4}) = T(N, N') \cdot \big(\mathsf{size}(\text{Sub-protocol 3})\big) + N(4+2K)\,,$$

$$\mathsf{preDepth}(\text{Protocol 4}) = 3\,, \quad \mathsf{onDepth}(\text{Protocol 4}) = 3 + 12\lceil \log_{N'} N \rceil\,.$$

Protocol 6. The costs of this protocol were analyzed in Section 4.2. After plugging in them the concrete costs of the interfaces that it invokes, and after changing the notation for the upper bound that we used in Section 4.2 from Q to 2^B, we arrive at the following costs:

$$\begin{aligned}\mathsf{size}(\text{Protocol } 6) &= N(2^B-1)+(2^B-1)(5NK+8N-K-2)+N(3+2K)+NK\\ &= (2^B-1)(5NK+9N-K-2)+N(3+3K)\,,\end{aligned}$$

$$\mathsf{preDepth}(\text{Protocol } 6) = 3\,,\quad \mathsf{onDepth}(\text{Protocol } 6) = 1+4+2+1 = 8\,.$$

Protocol 7. The concrete costs of this protocol, as implied by the analysis in Sections 5.1 and 7.1, are as follows:

$$\begin{aligned}\mathsf{size}(\text{Protocol } 7) &= BN+3BN(2+K)+B(5NK+8N-K-2)+NK\\ &= BN(15+8K)+K(N-B)-2B\,,\end{aligned}$$

$$\mathsf{preDepth}(\text{Protocol } 7) = 3\,,\quad \mathsf{onDepth}(\text{Protocol } 7) = 2+6B\,.$$

Protocol 8. Protocol 8 involves the following parameters: d is a tunable parameter that determines the number base $Q = 2^d$ in which the inputs are represented, and $D = B/d$ is the number of digits in the representation. Its concrete costs, as implied by the analysis in Sections 5.2 and 7.1, are as follows:

$$\begin{aligned}\mathsf{size}(\text{Protocol } 8) &= ND(Q-1)+ND(2Q-1)(2+K)\\ &\quad +ND\,(5K(Q-1)+8(Q-1)-K-2)\\ &\quad +D(Q-1)(5NK+8N-K-2)+NK\\ &= D(Q-1)(N(12K+21)-K-2)+NK\,,\end{aligned}$$

$$\mathsf{preDepth}(\text{Protocol } 8) = 3\,,\quad \mathsf{onDepth}(\text{Protocol } 8) = 1+D(3+4+4)+1 = 11D+2\,.$$

Protocol 11. Cast in our ABB terminology, such a protocol starts with N calls to PrivateInput; the main loop consists of (up to) B iterations where each iteration includes $2N$ calls to Compare (for computing the bits $1_{x_n<M}$ and $1_{x_n>M}$, where x_n is one of the inputs and M is the current guess for the value of the maximum) followed by two calls to Or of N bits (note that Or and And have similar costs). That final computation entails N calls to Equality, followed by N calls to PublicOutput. Hence,

$$\mathsf{size}(\text{Protocol } 11) = N\cdot[\textsc{PrI}]+2B\cdot(N\cdot[\textsc{Com}]+[\textsc{Or}]_N)+N\cdot[\textsc{Equ}]+N\cdot[\textsc{PuO}]\,,$$

$$\mathsf{preDepth}(\text{Protocol } 11) = \max([\textsc{PrI}],[\textsc{Com}],[\textsc{Or}]_N,[\textsc{Equ}],[\textsc{PuO}])\,,$$

and

$$\mathsf{onDepth}(\text{Protocol } 11) = [\textsc{PrI}]+B\cdot([\textsc{Com}]+[\textsc{Or}]_N)+[\textsc{Equ}]+[\textsc{PuO}]\,.$$

The concrete costs for the setting considered in Section 7 are therefore:

$$\begin{aligned}\mathsf{size}(\text{Protocol 11}) &= N + 2B\cdot(N\cdot(2B + 5\log B + 23) + (5NK + 8N - K - 2)) + \\ &\quad N\cdot(3+2K) + NK \\ &= NB(4B + 10\log B + 10K + 62) + N(3K+4) - 2B(K+2)\,,\end{aligned}$$

$$\mathsf{preDepth}(\text{Protocol 11}) = \max(0,0,3,1,0) = 3\,,$$

$$\mathsf{onDepth}(\text{Protocol 11}) = 1 + B\cdot(4+4) + 2 + 1 = 8B + 4\,.$$