Distributed Protocols for Oblivious Transfer and Polynomial Evaluation

Aviad Ben Arie[0009−0002−9680−4494] and Tamir Tassa[0000−0001−9681−8824]

The Open University of Israel, Ra'anana, Israel
tamirta@openu.ac.il

Abstract. A secure multiparty computation (MPC) allows several parties to compute a function over their inputs while keeping their inputs private. In its basic setting, the protocol involves only parties that hold inputs. In *distributed* MPC, there are also external servers who perform a distributed protocol that executes the needed computation, without learning information on the inputs and outputs. Here we propose distributed protocols for several fundamental MPC functionalities. We begin with a Distributed Scalar Product (DSP) protocol for computing scalar products of private vectors. We build upon DSP in designing various protocols for Oblivious Transfer (OT): k-out-of-N OT, Priced OT, and Generalized OT. We also use DSP for Oblivious Polynomial Evaluation (OPE) and Oblivious Multivariate Polynomial Evaluation (OMPE). All those problems involve a sender and a receiver that hold private vectors and they wish to compute their scalar product. However, in each of these problems the receiver must submit a vector of a specified form. Hence, a crucial ingredient in our protocols is a sub-protocol for validating that the receiver's vector complies with the relevant restrictions, without learning anything else on that vector. Therefore, while previous studies presented distributed protocols for 1-out-of-N OT and OPE, our protocols are the first ones that are secure against malicious receivers. Our distributed protocols for the other OT variants and for OMPE are the first ones that handle such problems. Our protocols offer information-theoretic security, under the assumption that the servers are semi-honest and have an honest majority, and they are very efficient.

Keywords: Multiparty Computation · Distributed Protocols · Oblivious Transfer · Oblivious Polynomial Evaluation.

1 Introduction

Secure multiparty computation (MPC) [35] is a central field of study in cryptography that aims at designing methods for several parties to jointly compute some function over their inputs while keeping those inputs private. In the basic setting of MPC, there are n mutually distrustful parties, $P_1, \ldots, P_n$, that hold private inputs, $x_1, \ldots, x_n$, and they wish to compute some joint function on their inputs, $f(x_1, \ldots, x_n)$. (The function can be sometimes multi-valued and issue different outputs to different designated parties.) No party should gain any information

on other parties' inputs, beyond what can be inferred from their own input and the output.

Typically, the only parties that participate in the protocol are those that hold the inputs or those who need to receive the outputs. However, some studies considered a model of computation that is called *the mediated model* [2,3,11,16,29,32,12], *the client-server model*, [6,10,18,27], or *the distributed model* [4,8,9,22,24,25]. Protocols in that model involve also external *servers* (or *mediators*), $M_1, \ldots, M_D$, $D \geq 1$, to whom the parties outsource some of the needed computations. The servers perform the computations while remaining oblivious to the private inputs and outputs. It turns out that such a distributed model of computation offers significant advantages: it may facilitate achieving the needed privacy goals; it does not require the parties to communicate with each other (a critical advantage in cases where the parties cannot efficiently communicate among themselves, or do not even known each other); in some settings it reduces communication costs; and it allows the parties, that may run on computationally-bounded devices, to outsource costly computations to dedicated servers [29].

In this work we focus on basic MPC problems that involve two ($n = 2$) parties, Alice (the sender) and Bob (the receiver), and propose distributed MPC protocols for their solution. In each of the studied problems, Alice's and Bob's private inputs may be encoded as vectors in a vector space over a finite field $\mathbb{Z}_p$; specifically, $\mathbf{a} = (a_1, \ldots, a_N) \in \mathbb{Z}_p^N$ is Alice's private vector and $\mathbf{b} = (b_1, \ldots, b_N) \in \mathbb{Z}_p^N$ is Bob's, for some integer N. Alice and Bob delegate to a set of $D > 2$ servers, $M_1, \ldots, M_D$, secret shares in their private vectors. Subsequently, the servers perform a multiparty computation on the received secret shares in order to validate the legality of the inputs, if the problem at hand dictates rules by which the input vectors must abide. If the inputs were validated, the servers proceed to compute secret shares in the required output and then they send those shares to Alice and/or Bob who use those shares in order to reconstruct the required output. The computational burden on Alice and Bob is thus reduced to secret sharing computations in the initial and final stages.

Our contribution. We begin by discussing the generic problem of scalar product, in which the required output is the scalar product, $\mathbf{a} \cdot \mathbf{b}$, of the two private input vectors [13,14,34]. We propose a simple protocol in which Alice and Bob only perform secret sharing computations while the servers perform only local computations, without needing to communicate among themselves. Our distributed scalar product protocol is then used in the subsequent problems that we tackle.

Next, we consider the problem of oblivious transfer (OT) [15,26], which is a fundamental building block in MPC [19] and in many application scenarios such as Private Information Retrieval (PIR) [7]. We consider several variants of OT: 1-out-of-N OT [1,20,21,23], k-out-of-N OT [5], Priced OT [1], and Generalized OT [17,30]. While several previous studies proposed distributed protocols for 1-out-of-N OT, $N \geq 2$, ours is the first one that does not rely on Bob's honesty. Specifically, while previous distributed 1-out-of-N OT protocols enabled Bob to learn any single linear combination of Alice's N secret messages, our protocol

restricts Bob to learning just a single message, as mandated in OT (see our discussion in Section 7). As for the other OT variants that we consider, we are the first to propose distributed protocols for their solution.

Then we deal with the problem of Oblivious Polynomial Evaluation (OPE) [23,33]. Here, Alice holds a private uni- or multi-variate polynomial $f(\cdot)$ and Bob holds a private value α. The goal is to let Bob have $f(\alpha)$ so that Alice learns nothing on α while Bob learns nothing on f beyond what is implied by α and $f(\alpha)$. Here too, while existing distributed OPE protocols allow Bob to learn any single linear combination of f's coefficients (and thus amount to protocols of distributed scalar product) ours is the first one that restricts Bob to learning only point values of f, at a point of his choice. We are also the first to propose a distributed protocol for OMPE — Oblivibious *Multivariate* Polynomial Evaluation.

Our OT and OPE protocols demonstrate the advantages that the distributed model offers. The delegation of computation to dedicated servers significantly simplifies computations that are typically more involved when Alice and Bob are on their own. The bulk of the computation is carried out by the servers, while Alice and Bob are active only in the initial and final stages, that are computationally lean. Another prominent advantage of the distributed model is that it enables carrying out all of the MPC problems that we consider even when Alice and Bob do not know each other and thus cannot communicate among themselves. In fact, Alice can complete her part in the protocol well before Bob starts his. For example, if Alice is a data custodian that holds some database, her private vector could hold decryption keys for the items in her database. The other party, Bob, can be any client that wishes to retrieve one of the items in that database, while keeping Alice oblivious of his choice, which is encoded in his private vector. Alice and Bob can use our various OT protocols for that purpose. But as they need to communicate only with the servers, Bob may perform his retrieval long time after Alice had already uploaded all information relating to her database. Moreover, in such an application scenario there is a single Alice but many "Bobs". While other protocols (non-distributed or even distributed) require Alice to be responsive to each Bob, our protocols allow Alice to act just once, at the initialization stage, while from that point onward only the servers deal with each of the future requests of potential clients (Bob). Our distributed OMPE protocol also offers such advantages.

We assume that the servers are semi-honest and have an honest majority. Namely, the servers follow the prescribed protocol, but a minority of the servers may collude among themselves or with Alice or Bob and share their views in the protocol. Under these assumptions our protocols are information-theoretic secure and provide unconditional security to both Alice and Bob, even when some of the parties collude.

Outline of the paper. Section 2 provides the relevant cryptographic preliminaries and assumptions. In Section 3 we describe our distributed scalar product protocol. Section 4 is devoted to the various distributed OT protocols. In Section 5 we present the OMPE protocol. We report experimental results in Section 6.

In Section 7 we review the prior art on distributed OT and OPE protocols and compare those protocols to ours. We conclude in Section 8.

2 Preliminaries

Secret sharing. The main idea in our protocols for solving the various MPC problems discussed herein is to use secret sharing. Alice and Bob distribute among the D servers, $M_1, \ldots, M_D$, shares in each entry of their private vectors, using t-out-of-D Shamir's secret sharing scheme [28], with

$$t = \lfloor (D+1)/2 \rfloor . \tag{1}$$

(Hereinafter we shall refer to such sharing as (t, D)-sharing.) Namely, Alice generates for each entry a_n, $n \in [N] := \{1, \ldots, N\}$, a polynomial $f_n^A(x) = a_n + \sum_{i=1}^{t-1} \alpha_i x^i$, where α_i are secret random field elements, and then she sends to M_d the value $[a_n]_d := f_n^A(d)$, $d \in [D] := \{1, \ldots, D\}$. Bob acts similarly. The servers then execute some distributed computation on the received shares in order to arrive at secret shares in the needed output. At the end, they distribute to Alice and/or Bob shares in the desired output from which Alice and/or Bob may reconstruct that output. The underlying field $\mathbb{Z}_p$ is selected so that p is larger than all values in the underlying computation.

Computing arithmetic expressions in shared secrets. In our protocols we will need to securely compute arithmetic expressions of shared secrets, where the expressions are degree two polynomials in the secrets (namely, they are sums of addends, each involving at most one multiplication of two secrets). We proceed to describe how we execute such computations.

First, we recall that secret sharing is affine in the following sense: if s_1 and s_2 are two secrets that are independently (t, D)-shared among $M_1, \ldots, M_D$, and a, b, c are three public field elements, then the servers can compute shares in $as_1 + bs_2 + c$. Specifically, if $[s_i]_d$ is M_d's share in s_i, $i = 1, 2$, $d \in [D]$, then $\{a[s_1]_d + b[s_2]_d + c : d \in [D]\}$ is a proper (t, D)-sharing of $as_1 + bs_2 + c$.

We turn to discuss the multiplication of shared secrets. Assume that the servers hold (t, D)-shares in s_i, $i = 1, 2$, where M_d's share in s_i is $[s_i]_d$. Assume that each server M_d, $d \in [D]$, multiplies the two shares that he holds and gets $c_d = [s_1]_d[s_2]_d$. It is easy to see that the set $\{c_d : d \in [D]\}$ is a $(2t-1, D)$-sharing of $s_1 s_2$. Therefore, the servers can recover $s_1 s_2$ by computing $c_d = [s_1]_d[s_2]_d$, then interpolate a polynomial F of degree $2t-2$ based on $\{c_1, \ldots, c_D\}$, and consequently infer that $s_1 s_2 = F(0)$. For simplicity, we will assume hereinafter that D is odd, in which case $2t - 1 = D$. Hence, $\{c_d = [s_1]_d[s_2]_d : d \in [D]\}$ constitute a (D, D)-sharing in $s_1 s_2$.

Scrambling shares. In some cases we shall perform the above described multiplication procedure when s_1 and s_2 are related (specifically, when $s_2 = s_1 - 1$). In such cases, the above described practice is problematic since each server M_d would need to expose to his peers the product of his secret shares $[s_1]_d[s_2]_d$, and due to the known relation between s_1 and s_2, that product of shares may reveal

information on $[s_1]_d$ and $[s_2]_d$, and consequently also information on the value of s_1 and s_2.

To avoid such potential information leakage, the servers perform a *scrambling* of the shares $\{c_1, \ldots, c_D\}$, in the sense that they generate a new random set of shares $\{c'_1, \ldots, c'_D\}$ that are also (D, D)-shares in $s_1 s_2$. They do that in the following manner. Each server M_d, $d \in [D]$, generates a random (D, D)-sharing of 0 and distributes the resulting shares to all servers. Subsequently, each server adds up the zero shares that he had received from all D servers. As a result, the mediators will hold (D, D)-shares of 0, denoted $\{[0]_1, \ldots, [0]_D\}$, where each share distributes uniformly in $\mathbb{Z}_p$. Finally, each server M_d sets $c'_d = c_d + [0]_d$, $d \in [D]$. Clearly, $\{c'_1, \ldots, c'_D\}$ are also (D, D)-shares in $s_1 s_2$, and their values do not leak any information on the original shares in s_1 and s_2.

We note that it is essential to generate a new set of zero shares, $[0]_d$, $d \in [D]$, for each operation of scrambling. However, it is possible to prepare such shares offline, before running the protocol in which scrambling is needed.

Security assumptions. The servers are assumed to be semi-honest, i.e., they follow the prescribed protocol, but try to extract from their view in the protocol information on the private inputs. We also assume them to have an *honest majority*, in the sense that if some of them are corrupted by a malicious adversary, their number is smaller than $t = \lfloor (D+1)/2 \rfloor$ (Eq. (1)). Hence, our protocols are immune against a coalition of up to $t-1$ servers who collude among themselves or with Alice or Bob.

3 Distributed Scalar Product

Here we deal with the following MPC problem.

Definition 1. *(DSP) Assume that Alice has a private vector* $\mathbf{a} = (a_1, \ldots, a_N) \in \mathbb{Z}_p^N$*, and Bob has a private vector* $\mathbf{b} = (b_1, \ldots, b_N) \in \mathbb{Z}_p^N$*. They wish to compute their scalar product* $\mathbf{a} \cdot \mathbf{b}$ *without revealing any other information on their private vectors.*

Protocol 1 solves that problem. In the first loop (Lines 1-3), Alice and Bob distribute to the servers (t, D)-shares in each entry of their vectors. Then, each server M_d computes a (D, D)-share in $a_n \cdot b_n$ for each $n \in [N]$, and subsequently he computes a (D, D)-share in the scalar product into s_d (Line 5). He then sends that share to Alice and Bob (Line 6). So now Alice and Bob have a full set of (D, D)-shares in $\mathbf{a} \cdot \mathbf{b}$ so they can recover the needed scalar product by means of interpolation (Line 7).

The protocol is correct and secure as we state next.

Theorem 1. *Protocol 1 is correct and provides information-theoretic security to both Alice and Bob when all servers are semi-honest and have an honest majority. Moreover, a coalition of one of the parties (Alice or Bob) with any subset of* $t-1$ *servers does not yield any information beyond what is implied by that party's input and the output.*

Protocol 1: Distributed Scalar Product

Parameters: p - field size, N - the dimension of the vectors, D - number of servers, $t = \lfloor (D+1)/2 \rfloor$.

Inputs: Alice has a private vector $\mathbf{a} = (a_1, \ldots, a_N) \in \mathbb{Z}_p^N$, Bob has a private vector $\mathbf{b} = (b_1, \ldots, b_N) \in \mathbb{Z}_p^N$.

1 **forall** $n \in [N]$ **do**
2 | Alice sends to M_d, $d \in [D]$, a (t, D)-share in a_n, denoted $[a_n]_d$.
3 | Bob sends to M_d, $d \in [D]$, a (t, D)-share in b_n, denoted $[b_n]_d$.
4 **forall** $d \in [D]$ **do**
5 | M_d computes $s_d \leftarrow \sum_{n \in [N]} ([a_n]_d \cdot [b_n]_d)$.
6 | M_d sends s_d to Alice and Bob.
7 Alice and Bob use $\{s_1, \ldots, s_D\}$ to reconstruct $\mathbf{a} \cdot \mathbf{b}$.

Output: Alice and Bob get $\mathbf{a} \cdot \mathbf{b}$.

Due to the page limitation we omit all proofs and will provide them in the full version of this paper.

4 Distributed Oblivious Transfer

In this section we consider several variants of the Oblivious Transfer (OT) protocol. We begin with the basic variants of 1-out-of-N and k-out-of-N OT in Section 4.1. We then discuss Priced OT (Section 4.2). The case of Generalized OT is introduced in Section 4.3; the detailed discussion is deferred to the full version of this paper.

4.1 *k*-out-of-*N* Oblivious Transfer

The problem that we consider here is the following:

Definition 2. *(OT_k^N) Assume that Alice has a set of N messages, $m_1, \ldots, m_N \in \mathbb{Z}_p$. Bob wishes to learn k of those messages, say $m_{j_1}, \ldots, m_{j_k}$, for some $j_1, \ldots, j_k \in [N]$. A k-out-of-N Oblivious Transfer (OT_k^N) protocol allows Bob to learn $m_{j_1}, \ldots, m_{j_k}$, and nothing beyond those messages, while preventing Alice from learning anything about Bob's selection.*

We begin by considering the case $k = 1$ and then we address the general case. The OT_1^N problem can be reduced to DSP (Section 3) if Alice sets $\mathbf{a} := (m_1, \ldots, m_N)$ and Bob sets $\mathbf{b} := \mathbf{e}_j$ (the unit vector that consists of $N-1$ zeros and a single 1 in the jth entry, where j is the index of the message that Bob wishes to retrieve). However, the DSP protocol cannot be executed naïvely, since Bob may cheat and send to the servers shares in a vector that is not a unit vector and, consequently, he may obtain some linear combination of the messages, and not just a single message as dictated by the OT definition. Such an abuse of the protocol may enable Bob in some cases to learn more than just one message. For example, if Bob happens to know that m_1 belongs to some one-dimensional

subspace of $\mathbb{Z}_p^N$ while m_2 belongs to another one-dimensional subspace of $\mathbb{Z}_p^N$, then by choosing to learn the linear combination $m_1 + m_2$ he will be able to infer both m_1 and m_2. To that end, the DSP protocol can be executed only after the server apply some preliminary validation protocol:

Definition 3. *(DVV) Assume that the servers $M_1, \ldots, M_D$ hold (t, D)-shares in a vector $\mathbf{v} \in \mathbb{Z}_p^N$. Let W be a subset of $\mathbb{Z}_p^N$. A Distributed Vector Validation (DVV) protocol is a protocol that the servers may execute on their shares that outputs 1 if $\mathbf{v} \in W$ and 0 otherwise, and reveals no further information on $\mathbf{v}$ in the case where $\mathbf{v} \in W$.*

In our case $W = \{\mathbf{e}_j : j \in [N]\}$. The servers can validate that $\mathbf{b} \in W$ by verifying the following two conditions: (1) $b_n \cdot (b_n - 1) = 0$ for all $n \in [N]$; and (2) $\sum_{n \in [N]} b_n = 1$. Indeed, the first condition implies that all entries in $\mathbf{b}$ are either 0 or 1, while the second condition ascertains that exactly one of the entries equals 1. Note that if the two conditions are verified, then the servers may infer that Bob's vector is legal, but nothing more than that, as desired. Namely, if Bob is honest then his privacy is fully protected. However, if Bob is dishonest and distributed shares in a vector $\mathbf{b} \notin W$, then the above described DVV protocol will reveal some additional information on $\mathbf{b}$; however, that is acceptable since by acting dishonestly Bob looses his right for privacy.

Protocol 2 implements those ideas. After Alice and Bob set their vectors and distribute shares in them to the servers (Lines 1-5), the servers validate Bob's vector for compliance with conditions 1 (Lines 6-12) and 2 (Lines 13-17). (The scrambling operation in Line 8 is as discussed in Section 2.) If Bob's vector was validated, they compute (D, D)-shares in the scalar product and send them to Bob so that he can recover the scalar product that equals his message of choice (Lines 18-21).

For a general $k > 1$, it is possible to solve OT_k^N by running Protocol 2 k times, with one exception: Alice needs to distribute shares in her vector only once (Lines 1 and 4 in Protocol 2). We proceed to describe another solution that is more efficient in terms of communication complexity.

Protocol 3 multiplies Alice's vector $\mathbf{a} := (m_1, \ldots, m_N)$ with the vector $\mathbf{b} = \sum_{i=1}^{k} \mathbf{e}_{j_i}$ where $1 \leq j_1 < \ldots < j_k \leq m$ are the indices of the k messages that Bob wishes to retrieve. But instead of computing their scalar product, $\sum_{n=1}^{N} a_n b_n$, the protocol computes shares in the products $a_n b_n$ for all $n \in [N]$ and sends them to Bob. Bob then uses the shares of $a_n b_n$ only for $n \in \{j_1, \ldots, j_k\}$ in order to recover the requested messages.

Here, the DVV sub-protocol consists of verifying two conditions: that $b_n \cdot (b_n - 1) = 0$ for all $n \in [N]$, and that $\sum_{n \in [N]} b_n = k$. The first condition implies that all entries in $\mathbf{b}$ are either 0 or 1, while the second condition ascertains that exactly k of the entries equal 1.

After Alice and Bob set their vectors and distribute shares in them to the servers (Lines 1-5), the servers validate Bob's vector for compliance with conditions 1 (Lines 6-12) and 2 (Lines 13-17). If Bob's vector was validated, they compute (D, D)-shares in each of the N products between the components of the

two vectors and send them to Bob (Lines 18-21) for him to recover the requested k messages (Lines 22-23).

Protocol 2: 1-out-of-N Oblivious Transfer

Parameters: p - field size, N - number of messages, D - number of servers, $t = \lfloor (D+1)/2 \rfloor$.

Inputs: Alice has $U = \{m_1, \ldots, m_N\}$; Bob has a selection index $j \in [N]$.

1 Alice sets $\mathbf{a} = (m_1, \ldots, m_N)$.
2 Bob sets $\mathbf{b} = \mathbf{e}_j$.
3 **forall** $n \in [N]$ **do**
4 | Alice sends to M_d, $d \in [D]$, a (t, D)-share in a_n, denoted $[a_n]_d$.
5 | Bob sends to M_d, $d \in [D]$, a (t, D)-share in b_n, denoted $[b_n]_d$.
6 **forall** $1 \leq n \leq N$ **do**
7 | Each M_d, $d \in [D]$, sets $c_d = [b_n]_d \cdot ([b_n]_d - 1)$.
8 | The servers perform scrambling of $(c_1, \ldots, c_D)$ and compute a new set of (D, D)-shares in $b_n \cdot (b_n - 1)$, denoted $(c'_1, \ldots, c'_D)$.
9 | Each M_d, $d \in [D]$, broadcasts c'_d.
10 | The servers use $(c'_1, \ldots, c'_D)$ in order to compute $\omega := b_n \cdot (b_n - 1)$.
11 | **if** $\omega \neq 0$ **then**
12 | | **Abort**
13 **forall** $d \in [D]$ **do**
14 | M_d computes $c_d \leftarrow \sum_{n \in [N]} [b_n]_d$.
15 The servers use any t shares out of $\{c_1, \ldots, c_D\}$ to compute $\omega := \sum_{n \in [N]} b_n$.
16 **if** $\omega > 1$ **then**
17 | **Abort**
18 **forall** $d \in [D]$ **do**
19 | M_d computes $s_d \leftarrow \sum_{n \in [N]} ([a_n]_d \cdot [b_n]_d)$.
20 | M_d sends s_d to Bob.
21 Bob uses $\{s_1, \ldots, s_D\}$ to reconstruct $\mathbf{a} \cdot \mathbf{b} = m_j$.

Output: Bob gets m_j.

Theorem 2. *Protocols 2 and 3 are correct and provide information-theoretic security to both Alice and an honest Bob when all servers are semi-honest and have an honest majority. Moreover, a coalition of one of the parties (Alice or Bob) with any subset of $t-1$ servers does not yield any information beyond what is implied by that party's input and the output.*

In the full version of this paper we describe an alternative 1-out-of-N Oblivious Transfer protocol that is also based on DSP. In that protocol, the DVV process is replaced by another mechanism that is based on an idea that was presented by Naor and Pinkas in [24] for their 1-out-of-2 OT protocol. The advantage in that protocol is that it does not require the servers to communicate with each other. However, on the down side, it enforces Alice to be responsive to any OT request of any client (Bob), as opposed to Protocol 2 in which Alice finishes her part in the initial phase.

Protocol 3: k-out-of-N Oblivious Transfer

Parameters: p - field size, N - number of messages, D - number of servers, $t = \lfloor (D+1)/2 \rfloor$.

Inputs: Alice has $U = \{m_1, \ldots, m_N\}$; Bob has selection indices $1 \leq j_1 < \ldots < j_k \leq N$.

Alice sets $\mathbf{a} = (m_1, \ldots, m_N)$.
Bob sets $\mathbf{b} = (b_1, \ldots, b_N)$, where $b_n = 1$ for $n \in \{j_1, \ldots, j_k\}$ and $b_n = 0$ otherwise.
forall $n \in [N]$ **do**
  Alice sends to M_d, $d \in [D]$, a (t, D)-share in a_n, denoted $[a_n]_d$.
  Bob sends to M_d, $d \in [D]$, a (t, D)-share in b_n, denoted $[b_n]_d$.
forall $1 \leq n \leq N$ **do**
  Each M_d, $d \in [D]$, sets $c_d = [b_n]_d \cdot ([b_n]_d - 1)$.
  The servers perform scrambling of $(c_1, \ldots, c_D)$ and compute a new set of (D, D)-shares in $b_n \cdot (b_n - 1)$, denoted $(c'_1, \ldots, c'_D)$.
  Each M_d, $d \in [D]$, broadcasts c'_d.
  The servers use $(c'_1, \ldots, c'_D)$ in order to compute $\omega := b_n \cdot (b_n - 1)$.
  if $\omega \neq 0$ **then**
    Abort
forall $d \in [D]$ **do**
  M_d computes $c_d \leftarrow \sum_{n \in [N]} [b_n]_d$.
The servers use any t shares out of $\{c_1, \ldots, c_D\}$ to compute $\omega := \sum_{n \in [N]} b_n$.
if $\omega \neq k$ **then**
  Abort
forall $d \in [D]$ **do**
  forall $n \in [N]$ **do**
    M_d computes $[c_n]_d \leftarrow [a_n]_d \cdot [b_n]_d$.
    M_d sends $[c_n]_d$ to Bob.
forall $n \in \{j_1, \ldots, j_k\}$ **do**
  Bob uses $\{[c_n]_1, \ldots, [c_n]_D\}$ to reconstruct $c_n = a_n \cdot b_n = m_n$.

Output: Bob gets $m_{j_1}, \ldots, m_{j_k}$.

4.2 Priced Oblivious Transfer

Consider a setting of OT in which each of Alice's messages has a weight and the retrieval policy allows Bob to learn any subset of messages in which the sum of weights does not exceed some given threshold. For example, if Alice holds a database of movies and each movie has a price tag, then if Bob had prepaid some amount, Alice wishes to guarantee that he retrieves movies of aggregated cost that does not exceed what he had paid, while Bob wishes to prevent Alice from knowing what movies he chose to watch.

Definition 4. *Let $U = \{m_1, \ldots, m_N\}$ be the set of messages that Alice has. Assume that each massage m_n has a weight $w_n \geq 0$, $n \in [N]$, and let $T > 0$ be some given threshold. Then a Priced OT protocol allows Bob to retrieve any subset $B \subseteq U$ for which $\sum_{m_n \in B} w_n \leq T$. Bob cannot learn any information on the messages in $U \setminus B$, while Alice has to remain oblivious of Bob's choice.*

We assume that the weights $w_1, \ldots, w_N$ are publicly known, since they represent information that is supposed to be known to all. The threshold T, on the other hand, represents the amount that Bob had paid and, therefore, it is private and should remain so.

Protocol 4 executes Priced OT. It coincides with Protocol 3 except for the second part of the DVV sub-protocol (Lines 13-17). If in Protocol 3 the servers obliviously verified in that part that $\sum_{n\in[N]} b_n \leq k$, then here it is necessary to obliviously verify that $\sum_{m_n\in B} w_n = \sum_{n\in[N]} w_n b_n \leq T$. (Recall that in Lines 6-12 in Protocol 3 we have already verified that $b_n \in \{0,1\}$, for all $n \in [N]$.) To enable that verification, the protocol starts by publishing the vector of weights (Line 1). Then, both Alice and Bob distribute to the servers (t, D)-shares in T (Lines 2-3) and then the servers verify that the two underlying thresholds equal, without recovering that threshold (Lines 4-7). Those steps are necessary in order to ascertain that Alice and Bob agree on the same value of the threshold, before using that value in the DVV sub-protocol. (Namely, Bob is ascertained that Alice did not provide a too low value of T while Alice is ascertained that Bob did not provide a too high value of T).

The core of the protocol is the execution of the OT_k^N protocol - Protocol 3 (Line 8). That protocol is executed as is except for the replacement of Lines 13-17 there with Sub-protocol 5. The sub-protocol begins with the servers computing (t, D)-shares in the difference $e := T - \sum_{n\in[N]} w_n b_n$ (Lines 1-2). Then, any subset of t servers can recover e (Line 3). Finally, if $e \neq 0$ the protocol aborts (Line 4), while otherwise it proceeds towards completing the transfer.

Note that Bob is allowed to retrieve any subset of messages of aggregated weight at most T. Sub-protocol 5, however, assumes that Bob had requested a subset of messages of aggregated weight that equals exactly T. Such an equality can be guaranteed as we proceed to describe. First, Bob can add to his list of requested messages additional redundant messages that he will ignore later on. By adopting such a practice, the difference $e = T - \sum_{n\in[N]} w_n b_n$ can be made a nonnegative number smaller than $\overline{w} := \max_{n\in[N]} w_n$. Assume that $\overline{w} < 2^\ell$, for some $\ell > 0$. Then Alice may add ℓ phantom messages $\hat{m}_i$, $0 \leq i < \ell$, with the weights 2^i, to her list of messages. Consequently, Bob will add to his list of requested messages also the subset of phantom messages of which the sum of weights equals exactly e. That way, the servers will always recover in Line 3 in Sub-protocol 5 the value 0.

The Case of Secret Weights Even though the weights of messages are typically public, it is possible to modify the protocol so that also the weights remain hidden from the servers. To do that, instead of publishing the vector of weights $\mathbf{w}$ (as done in Line 1 of Protocol 4), Alice would distribute to the servers (t, D)-shares in them. Let $[w_n]_d$ denote M_d's share in w_n, $d \in [D]$, $n \in [N]$. Then, in Sub-protocol 5, Line 2 will be replaced with $[e]_d \leftarrow [T]_d - \sum_{n\in[N]} [w_n]_d [b_n]_d$. As discussed in Section 2, the set $\{[e]_1, \ldots, [e]_D\}$ is a set of (D, D)-shares in e. The servers may use those shares in order to reconstruct $e = T - \sum_{n\in[N]} w_n b_n$. No further changes are required.

Protocol 4: Priced Oblivious Transfer

Parameters: p - field size, N - number of messages, D - number of servers, $t = \lfloor (D+1)/2 \rfloor$.

Inputs: Alice has $U = \{m_1, \ldots, m_N\}$, and corresponding weights $w_n \geq 0$, $n \in [N]$; Bob has a set of selection indices $j_1, \ldots, j_k \in [N]$; Alice and Bob have $T \geq 0$.

1 Alice publishes the vector of weights $\mathbf{w} = (w_1, \ldots, w_N)$.
2 Alice sends to M_d, $d \in [D]$, a (t, D)-share in T, denoted $[T]_d$.
3 Bob sends to M_d, $d \in [D]$, a (t, D)-share in T, denoted $[T']_d$.
4 **forall** $d \in [D]$ **do**
5 | M_d computes $[e]_d \leftarrow [T]_d - [T']_d$.
6 The servers use any t shares out of $\{[e]_1, \ldots, [e]_D\}$ to compute $e = T - T'$.
7 **if** $e \neq 0$ **then Abort**.
8 Alice, Bob and the servers execute Protocol 3 in which Lines 13-17 are replaced with Sub-protocol 5.

Output: Bob gets $\{m_{j_1}, \ldots, m_{j_k}\}$ iff $\sum_{i=1}^{k} w_{j_i} \leq T$.

Sub-protocol 5: Priced OT: verifying that $\sum_{i=1}^{k} w_{j_i} \leq T$.

1 **forall** $d \in [D]$ **do**
2 | M_d computes $[e]_d \leftarrow [T]_d - \sum_{n \in [N]} w_n [b_n]_d$.
3 The servers use any t shares out of $\{[e]_1, \ldots, [e]_D\}$ to compute $e = T - \sum_{n \in [N]} w_n b_n$.
4 **if** $e \neq 0$ **then Abort**.

As Protocol 4 coincides with Protocol 3 where only the DVV part is slightly modified, Theorem 2 applies also to that protocol, in both cases (public or secret weights).

4.3 Generalized Oblivious Transfer

Ishai and Kushilevitz [17] presented an extension of OT called *Generalized Oblivious Transfer* (GOT). While in OT_k^N Bob was restricted to learn any subset of at most k out of the N messages that Alice has, in GOT the policy is extended as described below.

Definition 5. *Let $U = \{m_1, \ldots, m_N\}$ be the set of messages that Alice has. An access structure is a collection of subsets of U, $\mathcal{A} \subseteq 2^U$, which is monotone decreasing in the sense that if $B \in \mathcal{A}$ and $B' \subset B$ then also $B' \in \mathcal{A}$.*

Bob is allowed to retrieve any subset of messages $B \subset U$ provided that $B \in \mathcal{A}$. As before, Bob cannot learn any information on messages in $U \setminus B$, while Alice must remain oblivious of Bob's selection. The retrieval policy can be determined by any access structure on the set of messages, e.g. a multipartite access structure [31], in which the set of messages is partitioned into distinct compartments and

the question of whether Bob may retrieve $B \subset U$ is determined only by the number of messages that B has in each of the compartments.

In the full version of this paper we present a protocol for that purpose that is based on the GOT protocol that was presented in [30], and it invokes the OT_k^N protocol, Protocol 3.

5 Oblivious Polynomial Evaluation

The oblivious polynomial evaluation problem was presented in [23], and was extended to the case of multivariate polynomials in [33]. We devise herein a distributed protocol for the multivariate problem.

We begin by defining multivariate polynomials (Definitions 6 and 7) and then define the corresponding MPC problem (Definition 8).

Definition 6. *(Monomial) Let $\mathbb{Z}_p$ be a finite field , $\mathbf{x} = (x_1, \ldots, x_k)$ be a k-dimensional vector over $\mathbb{Z}_p$ and $\mathbf{j} = (j_1, \ldots, j_k)$ be a k-dimensional vector of nonnegative integers. Then the monomial $\mathbf{x}^{\mathbf{j}}$ is defined as $\mathbf{x}^{\mathbf{j}} := \prod_{i=1}^{k} x_i^{j_i}$.*

Definition 7. *(Multivariate Polynomial) let $\mathbb{Z}_+^k := \{\mathbf{j} = (j_1, \ldots, j_k) : j_i \in \mathbb{Z}_+ = \{0, 1, 2, \ldots\} : 1 \le i \le k\}$ be the set of all k-tuples of nonnegative integers, and $\mathbb{Z}_+^{k,N}$ be the subset of $\mathbb{Z}_+^k$ consisting of all tuples of which the sum of components is at most N, i.e: $\mathbb{Z}_+^{k,N} := \{\mathbf{j} \in \mathbb{Z}_+^k : |\mathbf{j}| := \sum_{i=1}^{k} j_i \le N\}$. An N-degree k-variate polynomial $f(\mathbf{x})$ over the field $\mathbb{Z}_p$, where $\mathbf{x} = (x_1, \ldots, x_k) \in \mathbb{Z}_p^k$, is defined as:*

$$f(\mathbf{x}) = \sum_{\mathbf{j} \in \mathbb{Z}_+^{k,N}} a_{\mathbf{j}} \cdot \mathbf{x}^{\mathbf{j}}\,, \quad a_{\mathbf{j}} \in \mathbb{Z}_p\,. \tag{2}$$

Definition 8. *(OMPE) Assume that Alice has an N-degree multivariate polynomial $f(\mathbf{x}) = f(x_1, \ldots, x_k)$, while Bob has a point $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k) \in \mathbb{Z}_p^k$. They wish to enable Bob to learn $f(\boldsymbol{\alpha})$, and nothing else on f, while keeping Alice oblivious to $\boldsymbol{\alpha}$.*

OMPE can be solved by reducing it to DSP, with the needed prior validations. The vector that Alice will submit to the protocol consists of the coefficients of her polynomial, $\mathbf{a} = (a_{\mathbf{j}} : \mathbf{j} \in \mathbb{Z}_+^{k,N})$. The vector that Bob will submit to the protocol is the following:

$$\mathbf{b} = (b_{\mathbf{j}} : \mathbf{j} \in \mathbb{Z}_+^{k,N})\,, \quad \text{where} \quad b_{\mathbf{j}} := \boldsymbol{\alpha}^{\mathbf{j}}\,. \tag{3}$$

It is easy to see that the dimension of these vectors is $\binom{N+k}{k}$.

First, it is necessary to agree upfront on an ordering of $\mathbb{Z}_+^{k,N}$ so that in the scalar product between the two vectors, each power of $\boldsymbol{\alpha}$ will be multiplied by the corresponding polynomial coefficient. We suggest ordering the set $\mathbb{Z}_+^{k,N}$ by arranging its monomials into $N+1$ tiers, as follows. The 0th tier would be $T_0 := Z_+^{k,0}$, and then the nth tier, $n = 1, \ldots, N$, would be $T_n := Z_+^{k,n} \setminus Z_+^{k,n-1}$; namely,

Protocol 6: Oblivious Multivariate Polynomial Evaluation

Parameters: p - field size, k-number of variables, N - the degree of the secret polynomial f, D - number of servers, $t = \lfloor (D+1)/2 \rfloor$.

Inputs: Alice has a secret$N$-degree k-variate polynomial $f(\mathbf{x})$, Eq. (2); Bob has a secret point $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k) \in \mathbb{Z}_p^k$.

1 Alice sets $\mathbf{a} = (a_{\mathbf{j}} : \mathbf{j} \in \mathbb{Z}_+^{k,N})$, according to the ordering convention.

2 Bob sets $\mathbf{b} = (b_{\mathbf{j}} = \boldsymbol{\alpha}^{\mathbf{j}} : \mathbf{j} \in \mathbb{Z}_+^{k,N})$, according to the ordering convention.

3 **forall** $\mathbf{j} \in \mathbb{Z}_+^{k,N}$ **do**

4 Alice sends to M_d, $d \in [D]$, a (t, D)-share in $a_{\mathbf{j}}$, denoted $[a_{\mathbf{j}}]_d$.

5 Bob sends to M_d, $d \in [D]$, a (t, D)-share in $b_{\mathbf{j}}$, denoted $[b_{\mathbf{j}}]_d$.

6 **forall** $2 \leq n \leq N$ **do**

7 **forall** $\mathbf{j} \in T_n$ **do**

8 Select a monomial $\mathbf{h} \in T_{n-1}$ such that $\mathbf{j} = \mathbf{h} + \mathbf{e}_i$ for some $1 \leq i \leq k$, where $\mathbf{e}_i$ is the i-th unit vector.

9 The servers compute $\omega := b_{\mathbf{h}} \cdot b_{\mathbf{e}_i} - b_{\mathbf{j}}$.

10 **if** $\omega \neq 0$ **then**

11 **Abort**

12 **forall** $d \in [D]$ **do**

13 M_d computes $s_d \leftarrow \sum_{\mathbf{j} \in \mathbb{Z}_+^{k,N}} \left([a_{\mathbf{j}}]_d \cdot [b_{\mathbf{j}}]_d \right)$.

14 M_d sends s_d to Bob.

15 Bob uses $\{s_1, \ldots, s_D\}$ to reconstruct $\mathbf{a} \cdot \mathbf{b} = f(\boldsymbol{\alpha})$.

Output: Bob gets $f(\boldsymbol{\alpha})$.

the nth tier T_n consists of all monomials of degree exactly $n \in \{0, 1, \ldots, N\}$. The order within each tier would be lexicographical.

Protocol 6 starts with Alice and Bob setting their input vectors $\mathbf{a}$ and $\mathbf{b}$ in accord with the ordering convention (Lines 1-2). Then they distribute to the servers (t, D)-shares in them (Lines 3-5). Observe that the first entry in $\mathbf{b}$, i.e. $\mathbf{b_j}$ for $\mathbf{j} = (0, \ldots, 0)$, equals 1 (see Eq. (3)). Hence, in Line 5 for $\mathbf{j} = (0, \ldots, 0)$ Bob does not generate and distribute shares; instead, each server M_d, $d \in [D]$, sets $[b_{\mathbf{j}}]_d = 1$.

After completing the distribution of shares, the servers perform the relevant DVV sub-protocol in order to validate that the secret input vector $\mathbf{b}$ is of the form as in Eq. (3) (Lines 6-11). To that end we state the following lemma.

Lemma 1. *The vector* $\mathbf{b} = (b_{\mathbf{j}} : \mathbf{j} \in \mathbb{Z}_+^{k,N})$, *where* $\mathbf{b_j} = 1$ *for* $\mathbf{j} = (0, \ldots, 0)$, *is of the form as in Eq. (3) if and only if* $\omega = 0$ *in all stages of the validation loop in Lines 6-11 of Protocol 6.*

In the final stage of Protocol 6, the servers compute (D, D)-shares in the scalar product and send them to Bob (Lines 12-14) who uses them in order to recover the scalar product (Line 15).

Example. We illustrate the validation process when $k = 2$ and $N = 2$. Bob is expected to submit here vectors of the form

$$\mathbf{b} = (b_{(0,0)}, b_{(1,0)}, b_{(0,1)}, b_{(2,0)}, b_{(1,1)}, b_{(0,2)}) = (1, \alpha_1, \alpha_2, \alpha_1^2, \alpha_1\alpha_2, \alpha_2^2)\,.$$

Since the first entry is always 1, and the next two entries can be anything, validation is applied only on the last three entries — $b_{(2,0)}$, $b_{(1,1)}$, and $b_{(0,2)}$:

- To validate $b_{(2,0)}$, we observe that there is only one way to represent the multi-index $\mathbf{j} = (2,0)$ as a sum $\mathbf{h} + \mathbf{e}_i$, namely, $(2,0) = (1,0) + (1,0)$. Hence, the DVV sub-protocol checks whether $b_{(2,0)} = b_{(1,0)} \cdot b_{(1,0)}$. Therefore, validation of this entry succeeds if and only if $b_{(2,0)} = \alpha_1^2$.
- Similarly, $b_{(0,2)}$ is validated if and only if $b_{(0,2)} = \alpha_2^2$.
- To validate $b_{(1,1)}$, we observe that $\mathbf{j} = (1,1) = \mathbf{h} + \mathbf{e}_i$ with $\mathbf{h} = (1,0)$ and $\mathbf{e}_i = (0,1)$ or with $\mathbf{h} = (0,1)$ and $\mathbf{e}_i = (1,0)$. In either case, the DVV sub-protocol checks whether $b_{(1,1)} = b_{(1,0)} \cdot b_{(0,1)} = \alpha_1 \cdot \alpha_2$.

We conclude by noting that the security guarantees of Protocol 6 are as stated in Theorem 2.

6 Experiments

Implementation details. We implemented our protocols in Java on a Lenovo Ideapad Gaming 3 laptop, powered by an AMD Ryzen 7 5800H processor and 16GB of RAM. The operating system was Windows 11 64-bit, and the environment was Eclipse-Workspace. A 64-bit prime number p was chosen at random for the size of the underlying field $\mathbb{Z}_p$. To enable computations modulo such prime, we used the BigInteger Java class. The code is available at `https://github.com/b1086960/Distributed_OT_OPE`.

All experiments were conducted on randomly generated vectors (or sets of messages or polynomials). Each experiment was repeated ten times and the average runtimes for Alice, Bob and the servers are reported (where the runtimes for the servers are averaged over the ten runs as well as over the D servers). The standard deviation is omitted from the graphical display of our results since it is barely noticeable.

Results. In the first experiment we tested our basic protocol that solves DSP, Protocol 1. Figure 1 shows the runtimes for Alice and Bob and the average runtimes for the servers as a function of N (the dimension of the two vectors). The runtimes in Figure 1 grow linearly in N. Figure 2 displays those runtimes as a function of D. The runtimes for Alice and Bob grow quadratically in D since they need to perform D polynomial evaluations where the polynomial is of degree $t - 1 = O(D)$. The servers' runtime, on the other hand, is not affected by D and only slightly fluctuates randomly between 125 and 150 milliseconds for all tested values of D.

In the next experiment we tested Protocol 3 that solves the OT_k^N problem. Here we focus only on the servers, since Bob's computations in that protocol

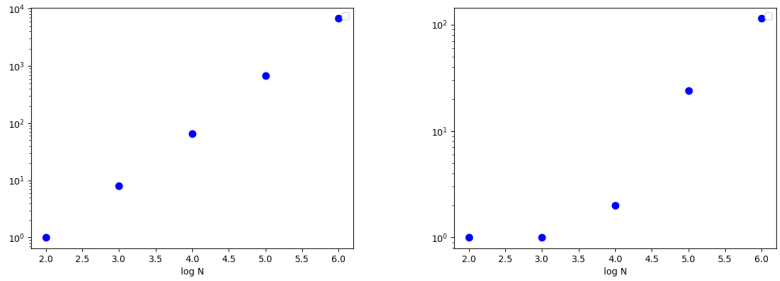

Fig. 1. Runtimes (milliseconds) for Protocol 1 (DSP), as a function of $\log_{10}(N)$, for $D = 7$. The left plot shows the runtimes for Alice and Bob; the right plot shows the average runtimes for the servers. The runtimes are presented on a logarithmic scale.

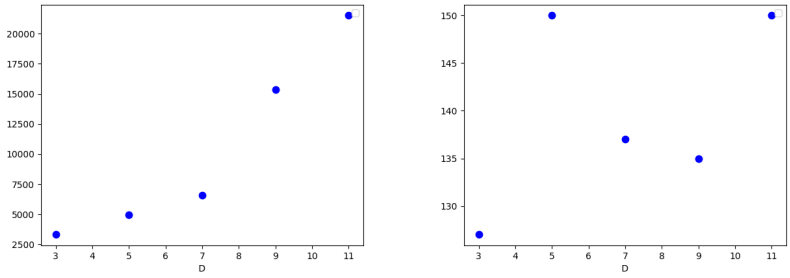

Fig. 2. Runtimes (milliseconds) for Protocol 1 (DSP), as a function of D, for $N = 10^6$. The left plot shows the runtimes for Alice and Bob; the right plot shows the average runtimes for the servers. The runtimes are presented on a linear scale.

are the same as in Protocol 1, while Alice's computations are the same as in the beginning of Protocol 1. The servers' runtimes are shown in Figure 3. The dependence on N is linear. As for D, while in Protocol 1 the servers' runtimes do not depend on D, here they do depend on D, linearly, due to the DVV part of the protocol. Their runtimes are not affected by k.

We turn our attention to Protocol 4 (Priced OT). Like in Protocol 3, we ignore the runtimes of Alice and Bob and focus on the servers' average runtime and demonstrate its linear dependence on N and on D, see Figure 4.

Finally, we consider Protocol 6 (OMPE). We ran that protocol with random polynomials of degrees $N \in \{5, 10, 20, 30, 40, 50\}$, where the number of variables was set to $k = 3$ — see Figure 5. The shown runtimes grow linearly with $\binom{N+k}{k}$, since that is the size of the two vectors in the scalar product.

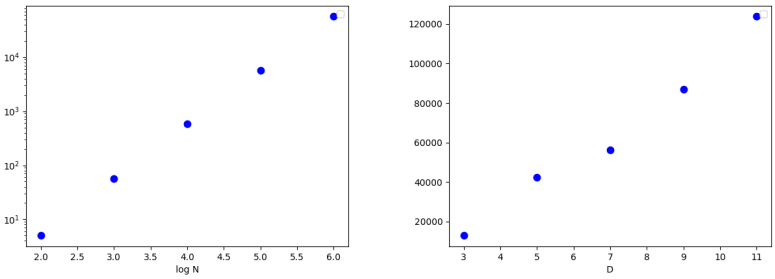

Fig. 3. Average runtimes (milliseconds) for the servers in Protocol 3 (OT_k^N). Left: runtimes, on a logarithmic scale, as a function of $\log_{10}(N)$, for $D = 7$ and $k = 10$. Right: runtimes as a function of D, for $N = 1000000$ and $k = 10$.

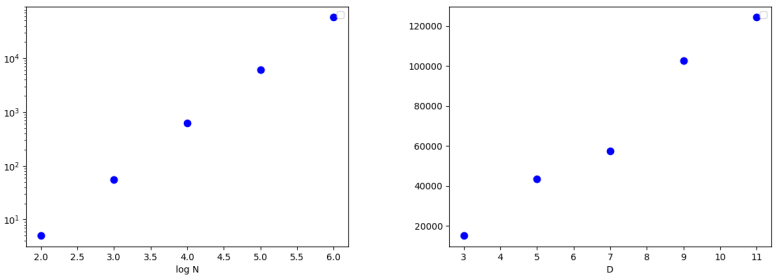

Fig. 4. Average runtimes (milliseconds) for the servers in Protocol 4 (Priced OT)). Left: runtimes as a function of N, for $T = 100$ and $D = 7$; the runtimes are presented on a logarithmic scale. Right: runtimes as a function of D, for $T = 100$ and $N = 1000000$.

7 Related Work

Naor and Pinkas [24] introduced the first version of a distributed OT. Their setting is similar to the one that we consider here: (a) apart from the sender (Alice) and the receiver (Bob) there are external servers that participate in the computation; (b) Alice sends information only to the servers and her role ends after doing so; (c) Bob can perform his part in a later time by communicating solely with the servers.

They considered OT_1^2: namely, Alice has m_1 and m_2, Bob has a selection index $j \in \{1, 2\}$, and the goal is to let Bob have m_j and nothing else, while Alice should remain oblivious of j. Their protocols are referred to as ℓ-out-of-D DOT_1^2, meaning that Bob has to communicate with ℓ out of the D servers in order to receive his message of choice.[1].

[1] In our discussion of related work we replace the original parameter notations with the ones that we used in the present work, for consistency and clarity.

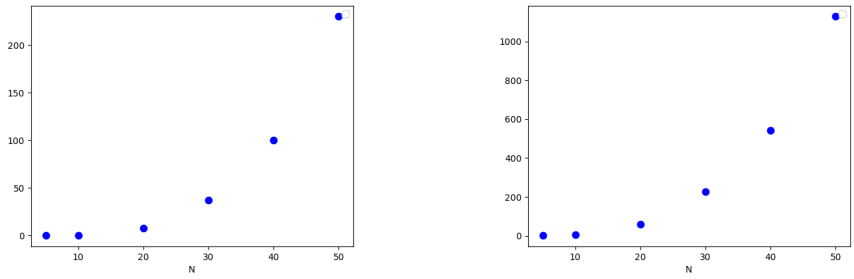

Fig. 5. Runtimes (milliseconds) for Protocol 6 (OMPE), as a function of N, the polynomial degree, for $k = 3$. Left: runtimes for Bob; right: average runtimes for the servers.

The two protocols that are proposed in [24] are based on secret sharing of some univariate polynomial. Specifically, Alice chooses a random bivariate polynomial $Q(x, y)$ that encodes m_1 and m_2, Bob chooses some random univariate polynomial $S(x)$ that encodes j, and then, by carefully selecting the degrees of those polynomials, they induce a univariate polynomial $R(x) = Q(x, S(x))$ of degree $\ell - 1$. The free coefficient in $R(x)$ is m_j and, consequently, Bob can get that value by obtaining the value of $R(x)$ in ℓ points. Bob does that by receiveing information from ℓ servers.

The first protocol uses a simple bivariate polynomial $Q(x, y)$. It suffers from two shortcomings: each server learns the difference $m_2 - m_1$ and, in addition, if a single server colludes with Bob, they obtain both of Alice's messages. The second protocol uses a more involved bivariate polynomial, that prevents the above described breach in Alice's privacy. However, that protocol still allows Bob to learn any linear combination of the two messages, rather than just m_1 or m_2. Later on they outline a manner which enforces Bob to learn just m_1 or m_2 but not any other linear combination of the two messages. The idea is to perform the protocol twice: in one execution Alice submits her two messages masked by random multipliers, $c_1 m_1$, and $c_2 m_2$; in the second execution Alice submits the two multipliers, c_1 and c_2. They then argue that if $m_1 \neq m_2$, such a course of action disables Bob from inferring any linear combination of m_1 and m_2 which is not one of the two messages.

Blundo et al. [4] generalized the protocols of [24] to distributed OT_1^N. In their generalization, Alice uses an N-variate polynomial. $Q(x, y_1, \ldots, y_{N-1})$ that encodes her N messages, $m_1, \ldots, m_N$. Bob, on the other hand, encodes his index j by $N - 1$ univariate polynomials, $Z_1, \ldots, Z_{N-1}$. Those polynomials implicitly induce a univariate polynomial of degree $\ell-1$, $R(x) = Q(x, Z_1(x), \ldots, Z_{N-1}(x))$, such that $R(0) = m_j$. As in [24], Bob contacts ℓ servers in order to get ℓ point values of R that enable him to recover $R(0) = m_j$. They showed that any coalition of up to $\ell - 1$ servers cannot obtain any information on j, and that any coalition of up to $\ell - 1$ servers with Bob cannot obtain any information on Alice's messages. However, their protocol has the same vulnerability as that of

[24]: each server learns the differences $m_n - m_1$ for all $1 \leq n \leq N$; and a coalition of Bob with a single server enables the recovery of *all* N messages.

Hence, the protocols of [24] and [4] are vulnerable to a collusion of Bob with just a single server. Blundo et al. defined the following privacy goal: a coalition of Bob with any subset of $\ell - 1$ servers should not be able to infer any information on Alice's messages, beyond the message that Bob had selected. They proved that such a goal cannot be achieved in a one-round DOT protocol.

Nikov et al. [25] presented an analysis of the ℓ-out-of-D DOT_1^N framework used in the above described studies. Namely, they considered protocols that involve a sender (Alice), a receiver (Bob) and D servers, through which Bob can retrieve a single message out of Alice's N messages by contacting ℓ of the D servers. They considered such a scheme to be (t, k)-secure if (a) any coalition of $t - 1$ servers cannot infer anything on Bob's selection index, and (b) a coalition of Bob with k corrupt servers does not yeild to Bob any further information. They then showed [25, Corollary 1] that such a scheme can exist iff $\ell \geq t + k$. They continued to demonstrate a construction of such a scheme with a minimal threshold of $\ell = t + k$. Later on, they considered settings in which not all servers enjoy the same level of trust and presented a DOT_1^N protocol in which Bob can recover his message of choice by contacting an authorized subset of servers, where the authorized subsets are defined by a general access structure.

We note that the protocols of [4,25] enable Bob to learn any single linear combination of Alice's messages, and not just a single message; hence, they implement only a weaker version of OT.

Corniaux and Ghodosi [9] took a different approach in their solution of the distributed OT_1^N problem. As opposed to the above described works, they allow the servers to communicate with each other, thus breaching out of the framework of one-round DOT. Their protocol is similar to our DOT_1^N protocol (Protocol 2): Alice distributes to the servers secret shares in her vector of messages, while Bob distributes secret shares in the binary vector that encodes his selection index. The requested message is the scalar product between those two private vectors. However, the protocol in [9] lacks the DVV part, which is at the heart of our Protocol 2 (Lines 6-15). Consequently, Bob can create any selection vector and hence can recover any linear combination of the messages $m_1, \ldots, m_N$. Hence, the protocol in [9] too does not implement OT but a weaker form of that problem. (We note that there are other technical differences between our Protocol 2 and the one in [9], e.g., the fact that we do not need to perform a transformation from one threshold scheme to another, as they do; we omit further details.)

The problem of OPE (Oblivious Polynomial Evaluation) was introduced by Naor and Pinkas in [23]. It is closely related to OT: here, too, Alice has a set of secrets and Bob is allowed to get a single linear combination of those secret while Alice should remain oblivious of his choice. While in OT the secrets are messages and the allowed linear combinations are the ones that consist of a single message, in OPE the secrets are the coefficients of a private polynomial, $f(x)$, and the allowed linear combinations are those that relate to a point value of that polynomial, $f(\alpha)$. In the OPE protocol of [23] Alice hides her secret polynomial

$f(x)$ in some bivariate polynomial while Bob hides his secret point α in some univariate polynomial. Those two polynomials induce a univariate polynomial $R(x)$ such that $R(0) = f(\alpha)$. Bob then learns $d_R + 1$ point values of R, where d_R is the degree of R, and then proceeds to recover $R(0)$. He does that by invoking $d_R + 1$ instances of 1-out-of-m OT, where m is a small security parameter.

We are interested here with distributed protocols for OPE. The first such protocol was introduced by Li et al. [22]. They suggested three protocols for that matter, which are based on secret sharing and polynomial interpolation. In the first and simplest method, Alice secret shares each of her polynomial coefficients among the servers, while Bob distributes secret shares in the corresponding powers of his selected point. The desired value is then obtained by computing the scalar product between the two shared vectors. The two subsequent versions of this basic protocol are designed in order to increase the immunity of the protocol to collusion between the servers and Bob. The protocols assume that all parties are semi-honest. Since Bob is also assumed to be semi-honest, Bob can submit to the protocol secret shares in any vector, not necessarily one of the form $(0, \alpha, \alpha^2, \ldots, \alpha^N)$ (where N is the degree of Alice's polynomial f). Hence, their protocols amount to protocols of distributed scalar product.

Cianciullo and Ghodosi [8] described another DOPE protocol that offers better security and complexity than the protocols of Li et al. [22]. Specifically, their protocol offers security for both Alice and Bob against collusion of up to $t - 1$ out of the D servers, for some threshold t that can be tuned by the degrees of the secret sharing polynomials that the protocol uses. Despite the advantages that their protocol offers with respect to that of Li et al. [22], it too does not restrict Bob to learning only point values of $f(x)$, as it allows Bob to learn any linear combination of f's coefficients. In addition, it requires Alice to communicate with Bob and generate a new set of secret shares per each request. Protocol 6 that we presented herein allows Alice to act just once and by thus serve an unlimited number of future queries of "Bobs"; it allows the computation only of point values of f; and it is the first protocol that is designed for multi-variate polynomials.

8 Conclusion

We presented here distributed MPC protocols for three fundamental MPC functionalities: scalar product, oblivious transfer (k-out-of-N, Priced, and Generalized OT), and oblivious (multivariate) polynomial evaluation (OMPE). While previous studies offered distributed MPC protocols for 1-out-of-N OT and for (univariate) OPE, ours are the first ones that consider malicious receivers and restrict them to receive only the outputs that the MPC problem dictates. To the best of our knowledge, our study is also the first one that suggests distributed MPC protocols for k-out-of-N OT, Priced OT, Generalized OT, and OMPE.

Our OT and OMPE protocols demonstrate the advantages that the distributed model offers: the existence of external servers enables much simpler and more efficient MPC protocols; it allows the MPC parties (the sender Alice

and the receiver Bob) to delegate the bulk of the computation to the dedicated servers; and it completely disconnects Alice from Bob so that they do not need to communicate with each other, or even to know each other or to be active at the same time. Moreover, in cases where the sender wishes to serve a multitude of receivers, she can perform her part just once, and from that point onward only the servers attend to any request of any future receiver.

When the servers are semi-honest and have an honest majority, our protocols are information-theoretic secure and provide unconditional security to both Alice and Bob, even when some of the parties collude. An interesting future research direction would be to strengthen our protocols in order to render them secure against malicious servers.

While OT and OPE can serve as building blocks for general MPC problems [19,22], it would be interesting to use the ideas presented here in order to develop distributed protocols for the following fundamental two-party MPC problems:

• Oblivious Function Evaluation (OFE): Alice has a function that is represented by a Boolean circuit and Bob has a suitable input binary vector. The goal is to let Bob learn the output of Alice's circuit over his input and nothing else, while Alice remains oblivious of Bob's input.

• Oblivious Automaton Evaluation (OAE): Alice has a deterministic finite or pushdown automaton $\mathcal{A}$ with an input alphabet Σ; Bob has a word $w \in \Sigma^*$. The goal is to let Bob learn whether w is a word that $\mathcal{A}$ accepts without learning any other information on $\mathcal{A}$, while Alice remains oblivious of w.

• Oblivious Turing Machine Evaluation (OTME): Alice has a Turing Machine M with an input alphabet Σ and Bob has a word $w \in \Sigma^*$. The goal is to let Bob know the output $M(w)$ without learning any other information on M, while Alice remains oblivious of w.

We believe that the distributed model can be most effective in designing solutions to such fundamental problems of multiparty computation as well as in practical problems that arise in privacy-preserving distributed computation.

References

1. William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, pages 119–135, 2001.
2. Joël Alwen, Jonathan Katz, Yehuda Lindell, Giuseppe Persiano, Abhi Shelat, and Ivan Visconti. Collusion-free multiparty computation in the mediated model. In *CRYPTO*, pages 524–540, 2009.
3. Joël Alwen, Abhi Shelat, and Ivan Visconti. Collusion-free protocols in the mediated model. In *CRYPTO*, pages 497–514, 2008.
4. Carlo Blundo, Paolo D'Arco, Alfredo De Santis, and Douglas R. Stinson. On unconditionally secure distributed oblivious transfer. *J. Cryptol.*, 20(3):323–373, 2007.
5. Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *CRYPTO*, pages 234–238, 1986.
6. Octavian Catrina and Florian Kerschbaum. Fostering the uptake of secure multiparty computation in e-commerce. In *ARES*, pages 693–700, 2008.
7. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
8. Louis Cianciullo and Hossein Ghodosi. Unconditionally secure oblivious polynomial evaluation: A survey and new results. *J. Comput. Sci. Technol.*, 37(2):443–458, 2022.
9. Christian L. F. Corniaux and Hossein Ghodosi. Scalar product-based distributed oblivious transfer. In *ICISC*, pages 338–354, 2010.
10. Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005.
11. Lihi Dery, Tamir Tassa, and Avishay Yanai. Fear not, vote truthfully: Secure multiparty computation of score based rules. *Expert Systems with Applications*, 168:114434, 2021.
12. Lihi Dery, Tamir Tassa, Avishay. Yanai, and Arthur Zamarin. Demo: A secure voting system for score based elections. In *CCS*, pages 2399–2401, 2021.
13. Wenliang Du and Mikhail J. Atallah. Privacy-preserving cooperative statistical analysis. In *ACSAC*, pages 102–110, 2001.
14. Wenliang Du and Justin Zhijun Zhan. A practical approach to solve secure multiparty computation problems. In *NSPW*, pages 127–135, 2002.
15. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
16. Alon Ben Horin and Tamir Tassa. Privacy preserving collaborative filtering by distributed mediation. In *RecSys*, pages 332–341, 2021.
17. Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *ISTCS*, pages 174–184, 1997.
18. Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, 2011. 272.
19. Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.
20. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS*, pages 818–829, 2016.
21. Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.

22. Hong-Da Li, Xiong Yang, Dengguo Feng, and Bao Li. Distributed oblivious function evaluation and its applications. *J. Comput. Sci. Technol.*, 19(6):942–947, 2004.
23. Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *STOC*, pages 245–254, 1999.
24. Moni Naor and Benny Pinkas. Distributed oblivious transfer. In *ASIACRYPT*, pages 205–219, 2000.
25. Ventzislav Nikov, Svetla Nikova, Bart Preneel, and Joos Vandewalle. On unconditionally secure distributed oblivious transfer. In *INDOCRYPT*, pages 395–408, 2002.
26. Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981.
27. Johannes Schneider. Lean and fast secure multi-party computation: Minimizing communication and local computation using a helper. In *SECRYPT*, pages 223–230, 2016.
28. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
29. Erez Shmueli and Tamir Tassa. Mediated secure multi-party protocols for collaborative filtering. *ACM Transactions on Intelligent Systems and Technology*, 11:1–25, 2020.
30. Tamir Tassa. Generalized oblivious transfer by secret sharing. *Des. Codes Cryptogr.*, 58(1):11–21, 2011.
31. Tamir Tassa and Nira Dyn. Multipartite secret sharing by bivariate interpolation. In *ICALP*, pages 288–299, 2006.
32. Tamir Tassa, Tal Grinshpoun, and Avishay Yanai. PC-SyncBB: A privacy preserving collusion secure DCOP algorithm. *Artificial Intelligence*, 297:103501, 2021.
33. Tamir Tassa, Ayman Jarrous, and Yonatan Ben-Ya'akov. Oblivious evaluation of multivariate polynomials. *J. Math. Cryptol.*, 7(1):1–29, 2013.
34. Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *SIGKDD*, pages 639–644, 2002.
35. Andrew C. Yao. Protocols for secure computation. In *FOCS*, pages 160–164, 1982.