Distributed Protocols for Oblivious Transfer and Polynomial Evaluation

Aviad Ben Arie Tamir Tassa*

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, MPC 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. We propose distributed protocols for several fundamental MPC functionalities. We begin with a Distributed Scalar Product (DSP) protocol. We then 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, both holding private vectors, and the goal is to let the receiver learn 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 the secure validation of the receiver's honesty. While previous studies presented distributed protocols for 1-out-of-N OT and OPE, ours are the first ones that are secure against dishonest receivers. Our distributed protocols for the other OT variants and for OMPE are the first ones that handle such problems. In addition, while previous art assumed semi-honest servers, we present protocols that are secure even when some of the servers are malicious. Our protocols offer information-theoretic security and they are very efficient.[1]

*Corresponding author. Email: tamir_tassa@yahoo.com

[1]A preliminary version of this paper appeared in Indocrypt 2023.

1 Introduction

Secure multiparty computation (MPC) [42] 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, 17, 22, 35, 38, 18], *the client-server model*, [11, 16, 24, 33], or *the distributed model* [8, 14, 15, 28, 30, 31]. 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 [35].

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 [19, 20, 40]. 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) [21, 32], which is a fundamental building block in MPC [25] and in many application scenarios such as Private Information Retrieval (PIR) [13]. We consider several variants of OT: 1-out-of-N OT [1, 26, 27, 29], k-out-of-N OT [9], Priced OT [1], and Generalized OT [23, 36]. 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 9). 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) [29, 39]. Here, Alice holds a private uni- or multivariate 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 consider first the case where 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. We then extend our discussion to include also malicious servers, and consider two adversarial assumptions: one in which the number of malicious servers, denoted c, is smaller than $\frac{D}{4}$ and another in which $c < \frac{D}{3}$. Our protocols are information-theoretic secure and provide unconditional security to both Alice and Bob.

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 analyze the communication complexity of all protocols in Section 6, and report experimental results in Section 7. In all of our discussion above we assumed that the servers are semi-honest and have an honest majority. In Section 8 we extend our discussion to the case in which c of the D servers are malicious; we consider the cases $c < \frac{D}{4}$ and $c < \frac{D}{3}$ and describe the necessary enhancements to our protocols that would render them information-theoretic secure in those cases. In Section 9 we review the prior art on distributed OT and

OPE protocols and compare those protocols to ours. We conclude in Section 10.

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 [34], 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 $\{c_1,\dots,c_D\}$, 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 $s_1 s_2$.

We recall that the BGW protocol [7] enables computing polynomial functions of shared secrets, where the polynomials can have any degree. The main ingredient in the BGW protocol is the reduction of the degree of the secret sharing polynomial of products of secrets back to the original degree. That is also the most costly part of that protocol. Since for our purposes it suffices to focus on degree two polynomials of shared secrets, we do not need to perform such a reduction of the secret sharing polynomial's degree and, thus, we do not invoke the BGW protocol.

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 $\{c_1,\dots,c_D\}$, in the sense that they generate a new random set of shares $\{c'_1,\dots,c'_D\}$ 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 $\{[0]_1,\dots,[0]_D\}$, 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, $\{c'_1,\dots,c'_D\}$ 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. For the sake of clarity we allocate the lion's part of our discussion to the case in which the servers are semi-honest (i.e., they follow the prescribed protocol, but try to extract from their view in the protocol information on the private inputs) and have an *honest majority* (i.e., if c of them collude, then $c < t = \lfloor (D+1)/2 \rfloor$.) We then extend our discussion to the case in which c of the servers are malicious. We consider two cases, one in which $c < \frac{D}{4}$ and one in which $c < \frac{D}{3}$.

Communication assumptions. We assume that the servers have an authenticated broadcast channel. Namely, each of the servers can broadcast a message to all its peers and the identity of the broadcaster can be authenticated.

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.*

Proof. The correctness of the protocol follows from our assumption that the servers are semi-honest and from the discussion in Section 2, namely that if $\{[s_i]_d : d \in [D]\}$ is a (t, D)-sharing of s_i, $i = 1, 2$, then $\{[s_1]_d \cdot [s_2]_d : d \in [D]\}$ is a (D, D)-sharing of $s_1 \cdot s_2$. The honest majority assumption ensures that the maximal number of colluding servers is $t - 1$, where t is the secret sharing threshold, Eq. (1). Hence, the information-theoretic security of Shamir's secret sharing scheme implies that Protocol 1 provides information-theoretic security for both Alice and Bob.

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 $\mathbf{b} = (b_1, \ldots, b_N) \in \mathbb{Z}_p^N$.

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 $d \in [D]$ **do**
- M_d computes $s_d \leftarrow \sum_{n \in [N]} ([a_n]_d \cdot [b_n]_d)$.
- M_d sends s_d to Alice and Bob.

Alice and Bob use $\{s_1, \ldots, s_D\}$ to reconstruct $\mathbf{a} \cdot \mathbf{b}$.

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

We now turn to prove the second claim. Assume that Bob colludes with $t-1$ servers in attempt to reap some additional information beyond what is implied by his input and the output. Bob's input $\mathbf{b}$ and the output $\alpha := \mathbf{a} \cdot \mathbf{b}$ reveal that Alice's vector is any vector in the $(N-1)$-dimensional affine space $V := \{\mathbf{a} \in \mathbb{Z}_p^N : \mathbf{a} \cdot \mathbf{b} = \alpha\}$. Assume that Bob colludes with $t-1$ servers, say $M_1, \ldots, M_{t-1}$. We need to prove that the information that the servers contribute does not yield any further information. The case $\mathbf{b} = \mathbf{0}$ is straightforward, since then Bob contributes nothing to the servers, and by the first part of the theorem, a coalition of $t-1$ servers learns no information on the private inputs. Hence, we focus on the case where $\mathbf{b} \neq \mathbf{0}$. We may assume, without loss of generality, that $b_N = 1$. We will show that in such a case, the information that Bob and $M_1, \ldots, M_{t-1}$ hold together allow a_n, $1 \leq n \leq N-1$, to be any value in $\mathbb{Z}_p$; that implies that $\mathbf{a}$ can be any vector in V.

Denote the secret sharing polynomial that Alice used for hiding a_n by $f_n(x) = \sum_{j=0}^{t-1} c_{n,j} x^j$, where $c_{n,0} = a_n$, $n \in [N]$. Then server M_d contributes the following N linear equations about the tN unknowns $\{c_{n,j} : 0 \leq j \leq t-1, n \in [N]\}$:

$$\sum_{j=0}^{t-1} c_{n,j} d^j = f_n(d) \,, \quad n \in [N] \,, \quad 1 \leq d \leq t-1 \,. \tag{2}$$

To those $(t-1)N$ equations in the tN unknowns Bob adds a single equation

$$\sum_{n=1}^{N} c_{n,0} b_n = \alpha \,. \tag{3}$$

In order to prove that the partial system of $(t-1)N+1$ linear equations in Eqs. (2)+(3) allows $(a_1, \ldots, a_{N-1}) = (c_{1,0}, \ldots, c_{N-1,0})$ to be any vector in $\mathbb{Z}_p^{N-1}$ we show that it is possible to add to that partial system the following $N-1$ equations,

$$c_{n,0} = \gamma_n \,, \quad 1 \le n \le N-1 \,, \tag{4}$$

and it will be solvable, for any arbitrary selection of γ_n, $1 \le n \le N-1$. Indeed, the resulting system of tN equations in tN unknowns has the form $M\mathbf{c} = \mathbf{r}$ where the matrix of coefficients, the vector of unknowns, and the right hand side vector are as shown below.

- M is an $N \times N$ matrix of $t \times t$ blocks that is structured as follows:

$$M := \begin{pmatrix} V_t & 0 & 0 & \cdots & 0 & 0 \\ 0 & V_t & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & V_t & 0 \\ 0 & 0 & 0 & \cdots & 0 & V_{t-1} \\ \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 & \cdots & \mathbf{b}_{N-1} & \mathbf{b}_N \end{pmatrix} . \tag{5}$$

The first $N-1$ rows in Eq. (7) consist of $t \times t$ blocks where V_t is the following Vandermonde matrix

$$V_t = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^{t-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t-1 & (t-1)^2 & \cdots & (t-1)^{t-1} \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix} .$$

The Nth row in Eq. (7) consists of $(t-1) \times t$ blocks, where V_{t-1} is the Vandermonde block of dimension $(t-1) \times t$,

$$V_{t-1} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^{t-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t-1 & (t-1)^2 & \cdots & (t-1)^{t-1} \end{pmatrix} . \tag{6}$$

Finally, the last row in M, Eq. (7), is a single row in which $\mathbf{b}_n$ is the t-dimensional row vector $(b_n, 0, \ldots, 0)$, $n \le [N]$. Hence, M is a lower triangular block matrix, where the blocks on its diagonal are V_t (recall that we assumed that $b_N = 1$).

• The vector of unknowns is $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_N)^T$, where $\mathbf{c}_n = (c_{n,0}, c_{n,1}, \ldots, c_{n,t-1})$, $n \in [N]$.

• The right hand side vector is $\mathbf{r} = (\mathbf{r}_1, \ldots, \mathbf{r}_N)^T$, where $\mathbf{r}_n = (f_n(1), \ldots, f_n(t-1), \gamma_n)$, for $1 \le n \le N-1$, while $\mathbf{r}_N = (f_N(1), \ldots, f_N(t-1), \alpha)$.

Since $\det M = (\det V_t)^N \neq 0$, the system of equations has a unique solution for any selection of $(\gamma_1, \ldots, \gamma_{N-1})$. Hence, the joint view of Bob and the $t-1$ servers still allows $\mathbf{a}$ to be any vector in $V = \{\mathbf{a} \in \mathbb{Z}_p^N : \mathbf{a} \cdot \mathbf{b} = \alpha\}$. $\Box$

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). Finally, we consider the case of Generalized OT in Section 4.3.

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 sometimes enable a malicious Bob 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 servers 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 $1 \le j_1 < \ldots < j_k \le m$ 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).

The communication complexity of Protocols 2 and 3 as well as of the protocols that we present later on (for Priced OT, Generalized OT, and OMPE) is discussed in Section 6.

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.*

Proof. The proof of the first part goes along the same lines as the proof of the first part of Theorem 1. The only distinction is that in Protocols 2 and 3 Bob may act dishonestly by submitting an illegal vector (while in Protocol 1 Bob, like Alice, is not restricted in any way). As we noted earlier, if Bob acts dishonestly the servers may infer information on the way he cheated. For example, if $b_n \notin \{0, 1\}$ then the value $b_n \cdot (b_n - 1)$ allows the servers to learn that b_n is one of two values. As stated in the theorem, the protocol preserves the privacy of Bob only when he is honest.

The proof of the second part for Protocol 2 is the same as the proof of the second part in Theorem 1. The proof for Protocol 3 is different, since the output that Bob receives is k out of the N messages, and not a single linear combination as in Protocols 1 and 2. Assume, without loss of generality, that the k messages that Bob selected are m_n, $N-k+1 \leq n \leq N$. In that case Bob and the $t - 1$ servers with whom he colluded (say, $M_1, \ldots, M_{t-1}$) can reduce the system of linear equations from tN unknowns to $t(N - k)$ unknowns, $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_{N-k})^T$, where $\mathbf{c}_n = (c_{n,0}, c_{n,1}, \ldots, c_{n,t-1})$, $1 \leq n \leq N - k$, are the coefficients of the share generating polynomials that Alice used for

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 $j \in [N]$.

Alice sets $\mathbf{a} = (m_1, \ldots, m_N)$.
Bob sets $\mathbf{b} = \mathbf{e}_j$.
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 $1 \le n \le N$ **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 $\omega > 1$ **then**
    Abort
forall $d \in [D]$ **do**
    M_d computes $s_d \leftarrow \sum_{n \in [N]} ([a_n]_d \cdot [b_n]_d)$.
    M_d sends s_d to Bob.
Bob uses $\{s_1, \ldots, s_D\}$ to reconstruct $\mathbf{a} \cdot \mathbf{b} = m_j$.

Output: Bob gets m_j.

hiding $m_n = c_{n,0}$, $1 \le n \le N - k$. The matrix of coefficients in the system of $(t-1)(N-k)$ linear equations that Bob and the servers hold in those

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 $1 \le j_1 < \ldots < j_k \le N$.

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 $1 \le n \le N$ **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 $m_{j_1}, \ldots, m_{j_k}$.

$t(N-k)$ unknowns is

$$M := \begin{pmatrix} V_{t-1} & 0 & 0 & \cdots & 0 \\ 0 & V_{t-1} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & V_{t-1} \end{pmatrix}, \tag{7}$$

where M consists of $(N-k) \times (N-k)$ blocks of dimension $(t-1) \times t$ and V_{t-1} is as in Eq. (6). That system can be augmented with the additional $N-k$ linear equations

$$c_{n,0} = \gamma_n \,, \quad 1 \le n \le N-k \,,$$

for any arbitrary selection of $(\gamma_1, \ldots, \gamma_{N-k})$, and the augmented system will have a unique solution. That proves that also here, a collusion of Bob with any $t-1$ servers does not yield additional information on the remaining messages. $\square$

4.1.1 An efficient DVV sub-protocol

The most costly part in Protocols 2 and 3 is that in which the servers verify that Bob's vector is a binary vector (Lines 6-12 in both protocols), as the communication cost of that part of the DVV depends linearly on N. Instead, the servers may perform an alternative computation as described in Sub-protocol 4.

First, they compute a joint random seed σ to any pseudorandom number generator of their choice (Lines 1-2). Then, each server computes locally a sequence of pseudorandom nonzero multipliers (Line 3). Because all servers have the same seed and the same pseudorandom number generator, they will all get the same sequence of multipliers $(c_1, \ldots, c_N)$. Then, each server computes ω_d (Line 4) which is a (D, D)-share in ω (Line 5). The servers recover ω from $\{\omega_d : d \in [D]\}$ (Line 5). If $\omega \neq 0$ then clearly there exists at least one index $n \in [N]$ for which $b_n \notin \{0, 1\}$, what implies that Bob's vector is not binary. In that case the servers abort the computation (Lines 6-7). Otherwise, the servers deduce that Bob's vector is binary, with probability at least $1 - 1/p$. Indeed, as the servers compute the pseudorandom multipliers c_n, $n \in [N]$, only after Bob had submitted shares in his input vector, then an illegal input vector would pass the test (i.e., would yield $\omega = 0$) in probability $1/p$.

While the DVV sub-protocol in Protocols 2 and 3 is fully secure, Sub-protocol 4 introduces a negligible probability of an error. However, the advantages in terms of communication costs and runtimes are dramatic, as we show later on.

In what follows, we refer to Protocols 2 and 3 in which Lines 6-12 are replaced with Sub-protocol 4 as Protocols 2* and 3*, respectively.

Sub-protocol 4: A verification that Bob's input vector is binary

Each M_d generates a random seed σ_d, $d \in [D]$.
The servers compute $\sigma = \sum_{d\in[D]} \sigma_d$.
Each M_d uses σ in order to compute $c_n \in \mathbb{Z}_p^*$, for all $n \in [N]$.
Each M_d broadcasts $\omega_d := \sum_{n\in[N]} c_n \cdot [b_n]_d \cdot ([b_n]_d - 1)$, $d \in [D]$.
The servers recover $\omega := \sum_{n\in[N]} c_n \cdot b_n \cdot (b_n - 1)$ from $\{\omega_d : d \in [D]\}$.
if $\omega \neq 0$ **then**
  Abort

4.1.2 A 1-out-of-N distributed OT protocol with non-interacting servers

In Appendix A 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 [30] 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.

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 $w_1, \dots, w_N$ 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 5 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 6. 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 6, 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 6 the value 0.

4.2.1 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 5), 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 6, 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 5: 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 $\quad$ 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 6.

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

As Protocol 5 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 [23] presented an extension of OT called *Generalized Oblivious Transfer* (GOT). As in the basic version of OT, Definition 2, we

Sub-protocol 6: Priced OT: verifying that $\sum_{i=1}^{k} w_{j_i} \leq T$.

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

consider a setting with two parties, Alice and Bob. Alice has a set of N messages, $m_1, \ldots, m_N$, that can be viewed as elements in a finite field $\mathbb{Z}_p$. Bob wishes to learn a subset of those messages, according to some retrieval policy. In OT_k^N, the policy restricted Bob to learn any subset of at most k messages. 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 $B' \in \mathcal{A}$. The basis of $\mathcal{A}$, denoted $\mathcal{A}^0$, is the collection of all maximal subsets in $\mathcal{A}$; namely, $B \in \mathcal{A}^0$ iff $B \in \mathcal{A}$ and for every $B \subsetneq B' \subseteq U$, $B' \notin \mathcal{A}$.*

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 the complement set of messages, $U \setminus B$, while Alice must remain oblivious to Bob's selection.

The distributed GOT protocol that we present here, Protocol 7, is inspired by the GOT protocol that was presented in [36], and it invokes the OT_k^N protocol, Protocol 3. Protocol 7 is designed for the case of uniform bases, namely, the case where all subsets in $\mathcal{A}^0$ have the same size, denoted k. The case of non-uniform bases can be reduced to the case of uniform bases as described in [36]. We refer the reader to [36] for a detailed description of the simple reduction.

Let us define the *monotone increasing closure* of $\mathcal{A}^0$ as follows:

$$\Gamma = \Gamma(\mathcal{A}^0) = \{C \subseteq U : \exists B \in \mathcal{A}^0, B \subseteq C\}\,. \tag{8}$$

The collection Γ is monotone increasing, in the sense that if $B \in \Gamma$ and $B \subset B' \subseteq U$, then also $B' \in \Gamma$. Let Σ be a secret sharing scheme that realizes Γ, in the following sense. It is a secret sharing scheme in which the set of participants is U, and the access structure is Γ. Given a secret s, the scheme Σ assigns a share s_n to each message $m_n \in U$, $n \in [N]$, so that the shares of any subset in Γ reveal s while the shares of any other subset reveal no information on s. We note that any monotone increasing access structure can be realized by a secret sharing scheme, see e.g. [4].

Protocol 7 starts with Alice selecting a secret $s \in \mathbb{Z}_p$ (Line 1). Then she computes corresponding shares in s according to the access structure $\Gamma(\mathcal{A}^0)$ (Line 2). Namely, if $B = \{m_{j_1}, \ldots, m_{j_k}\} \in \mathcal{A}^0$ is a set of messages that Bob is allowed to retrieve, the corresponding set of shares, $\{s_{j_1}, \ldots, s_{j_k}\}$, can be used to reconstruct s; otherwise, those shares reveal no information on s. Alice proceeds to distribute to the servers (t, D)-shares in the secret s (Line 3).

Afterwards, Alice distributes to the servers (t, D) shares in two private vectors: the vector of private shares (Line 4) and the vector of private messages (Line 5). The mediators keep both sets of shares for use later on. At this point Alice had completed her part in the protocol. The remainder of the protocol is executed by Bob and the servers.

Bob and the servers execute a k-out-of-N OT for the k shares corresponding to Bob's k selected messages (Line 6). Note that at this stage Bob only retrieves the shares $s_{j_1}, \ldots . s_{j_k}$ but not the actual messages. He then proceeds to reconstruct $s^B := s$ from those shares using the reconstruction function of the secret sharing scheme Σ (Line 7). Subsequently, Bob distributes to the servers (t, D)-shares in s^B (Line 8). The servers proceed to verify that $s^B = s^A = s$ without actually recovering s (Lines 9-10). If the difference $e = s^A - s^B$ is non-zero, then Bob failed to prove that he attempted retrieving an allowed subset of messages; in that case the protocol aborts (Line 11). Otherwise, the servers are convinced that Bob did submit a selection vector $\mathbf{b}$ that corresponds to an allowed subset of messages. Hence, they engage in the completion of the k-out-of-N OT, where this time they use the shares in the actual messages (Line 12). As a result, Bob retrieves his messages of choice.

If Bob acts honestly then the security guarantees of Protocol 7 are as those of Protocol 3, see Theorem 2. However, Bob may attempt guessing the value of $s \in \mathbb{Z}_p$. The probability of a successful guess is $1/p$; in that case Bob may be able to learn any subset of k messages. However, the probability of failing to guess s is overwhelming — $1 - 1/p$; if Bob fails in his cheating attempt then the servers would infer that he attempted cheating and could refuse to engage in further attempts.

Note that Alice performs secret sharing on s in two different places in

Protocol 7: Generalized Oblivious Transfer

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

Inputs: Alice has $U = \{m_1, \ldots, m_N\} \subset \mathbb{Z}_p$ and an access structure $\mathcal{A}$ on U, with a k-uniform basis $\mathcal{A}^0$; Bob has indices $1 \le j_1 < \ldots < j_k \le N$, where $B := \{m_{j_1}, \ldots, m_{j_k}\} \in \mathcal{A}^0$.

1 Alice selects uniformly at random a secret $s \in \mathbb{Z}_p$.

2 Alice computes shares $\{s_1, \ldots, s_N\} \subset \mathbb{Z}_p$ in s using a secret sharing scheme Σ that realizes the access structure $\Gamma(\mathcal{A}^0)$ on U.

3 Alice distributes to the servers (t, D)-shares in $s^A := s$; M_d's share is denoted $[s^A]_d$, $d \in [D]$.

4 Alice sets $\mathbf{a} = (s_1, \ldots, s_N)$ and then performs Lines 3+4 in Protocol 3.

5 Alice sets $\mathbf{a} = (m_1, \ldots, m_N)$ and then performs Lines 3+4 in Protocol 3.

6 Bob and the servers execute Lines 2,3+5,6-23 in Protocol 3, where the servers use the shares in $\mathbf{a} = (s_1, \ldots, s_N)$ that Alice had distributed to them in Line 4 above.

7 Bob recovers $s^B := s$ from $\{s_n : n \in \{j_1, \ldots, j_k\}\}$ using the reconstruction function of the secret sharing scheme Σ.

8 Bob distributes to the servers (t, D)-shares in the secret s^B that he had computed above; M_d's share is denoted $[s^B]_d$, $d \in [D]$.

9 M_d, for all $d \in [D]$, computes $[e]_d = [s^A]_d - [s^B]_d$.

10 The servers recover $e := s^A - s^B$ from any t shares out of $\{[e]_d : d \in [D]\}$.

11 **if** $e \neq 0$ **then** Abort.

12 Bob and the servers execute Lines 18-23 in Protocol 3 where the servers use the shares in $\mathbf{a} = (m_1, \ldots, m_N)$ that Alice had distributed to them in Line 5 above.

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

Protocol 7 and in two entirely different ways. In Line 2, Alice secret-shares s among the set of participants $U = \{m_1, \ldots, m_N\}$ where the access structure is Γ; the secret sharing scheme here is Σ. Later on, in Line 6, Alice secret-shares the same value s among the set of participants $\{M_1, \ldots, M_D\}$, i.e., the servers, where the access structure is a simple t-out-of-D threshold access structure and t is as defined in Eq. (1); the secret sharing scheme here is the standard Shamir threshold secret sharing scheme [34]. The purpose of the first secret sharing is to ensure that Bob can retrieve only subsets of k messages from $\mathcal{A}^0$. The purpose of the second secret sharing scheme is to enable the servers to verify that the value of s that Alice used equals the value of s that Bob sends to them, without actually knowing s. (In a simpler implementation, Alice could have sent the value of s to the servers, without secret sharing. But then if Bob is able to corrupt a single server, he could get from that server the value of s and then Bob would be able to learn any subset of k messages. That is something that we prevent in Protocol 7 which is secure under the assumption that the majority of servers are honest.)

Non-ideal access structures. We assumed that the access structure $\Gamma(\mathcal{A}^0)$, Eq. (8), is ideal in the sense that there exists a secret sharing scheme Σ that realizes it in which all secret shares $s_1, \ldots, s_N$ are taken from the same field $\mathbb{Z}_p$ as the secret s. In cases where $\Gamma(\mathcal{A}^0)$ is not ideal, or in cases where $\Gamma(\mathcal{A}^0)$ is ideal, but the selected secret sharing scheme Σ is not ideal[2], then the shares $s_1, \ldots, s_N$ cannot be taken from $\mathbb{Z}_p$. Assume that in such a case all shares can be embedded in $\mathbb{Z}_q$ for some prime $q \geq p$. Then Protocol 7 works exactly as described, where the execution of Protocol 3 with the vector $\mathbf{a} = (s_1, \ldots, s_N)$ will be executed over $\mathbb{Z}_q$.

A concluding remark. As noted earlier, Alice and Bob do not need to be active at the same time. In all OT variants that we considered (k-out-of-N, Priced and Generalized OT), Alice can complete her part and then go offline; only when the need arises, Bob can initiate the completion of the protocol. For example, if Alice is a data custodian that serves many "Bob" clients, Alice may complete her part and then let the servers attend to the request of any future client Bob.

4.3.1 Exemplifying GOT for compartmented message sets

Assume that the set of messages, $U = \{m_1, \ldots, m_N\}$, is split into r disjoint subsets, called compartments,

$$U = \bigcup_{i=1}^{r} U_i \,, \qquad U_i \cap U_j = \emptyset \,,\ 1 \leq i < j \leq r \,.$$

[2]It is possible that a non-ideal secret sharing scheme could be simpler and easier to implement than an equivalent ideal secret sharing scheme that realizes the same access structure.

Bob is allowed to retrieve messages only from one of those compartments. Hence, the access structure here is

$$\mathcal{A} = \{B \subset U : |B| \subseteq U_i \text{ for some } 1 \le i \le r\} .$$

The basis of this access structure is $\mathcal{A}^0 = \{U_i : 1 \le i \le r\}$, and its monotone increasing closure is

$$\Gamma = \Gamma(\mathcal{A}^0) = \{B \subset U : B \supseteq U_i \text{ for some } 1 \le i \le r\} . \tag{9}$$

The access structure in Eq. (9) is a simple case of a compartmented access structure [10, 37], namely, one in which the participants (messages) are split into disjoint compartments, and all participants within the same compartment play the same role in the access structure. The access structure Γ can be easily realized as follows. (What follows is the computation that Alice does in Line 2 of Protocol 7 in case her access structure is as described above.)

Alice selects a random secret $s \in \mathbb{Z}_p$ and then, for each compartment U_i, $1 \le i \le r$, she will assign to all messages in that compartment random secret shares that add up to s. Specifically, if $U_i = \{m_{j_h} : 1 \le h \le |U_i|\}$ then Alice selects uniformly at random $|U_i| - 1$ secret shares, $s_{j_h} \in \mathbb{Z}_p$, $1 \le h \le |U_i| - 1$, and then she sets $s_{j_{|U_i|}} = s - \sum_{h=1}^{|U_i|-1} s_{j_h} \bmod p$.

5 Oblivious polynomial evaluation

The oblivious polynomial evaluation problem was presented in [29], and was extended to the case of multivariate polynomials in [39]. 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:*

$$f(\mathbf{x}) = \sum_{\mathbf{j} \in \mathbb{Z}_+^{k,N}} a_{\mathbf{j}} \cdot \mathbf{x}^{\mathbf{j}}, \quad a_{\mathbf{j}} \in \mathbb{Z}_p. \tag{10}$$

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:

$$\mathbf{b} = (b_{\mathbf{j}} : \mathbf{j} \in \mathbb{Z}_+^{k,N}), \quad \text{where} \quad b_{\mathbf{j}} := \boldsymbol{\alpha}^{\mathbf{j}}. \tag{11}$$

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, 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 8 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. $\mathbf{b}_{\mathbf{j}}$ for $\mathbf{j} = (0, \ldots, 0)$, equals 1 (see Eq. (11)). 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. (11) (Lines 6-11). To that end we state the following lemma, which we prove in Appendix B.

Lemma 1. *The vector $\mathbf{b} = (b_{\mathbf{j}} : \mathbf{j} \in \mathbb{Z}_+^{k,N})$, where $\mathbf{b}_{\mathbf{j}} = 1$ for $\mathbf{j} = (0, \ldots, 0)$, is of the form as in Eq. (11) if and only if $\omega = 0$ in all stages of the validation loop in Lines 6-11 of Protocol 8.*

Protocol 8: 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. (10); 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 $f(\boldsymbol{\alpha})$.

In the final stage of Protocol 8, 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 8 are as stated in Theorem 2.

6 Communication complexity

Here we discuss the communication complexity of our protocols. We measure the complexity by counting field ($\mathbb{Z}_p$) elements, where each field element can be represented by $\lceil \log p \rceil$ bits,

We separate the overall communication traffic to three parts:

- Com_{AM}: Messages sent between Alice and the servers.

- Com_{BM}: Messages sent between Bob and the servers.

- Com_{MM}: Messages sent among the servers.

For Protocol 1 (DSP) we have $\text{Com}_{\text{AM}} = \text{Com}_{\text{BM}} = (N + 1)D$, since Alice and Bob send to each of the D servers shares in each of the N entries in their vectors and, at the end, each server sends a single share back to Alice and Bob. As in this protocol the servers do not communicate among themselves, we have $\text{Com}_{\text{MM}} = 0$.

The communication costs of Protocol 2 for the OT_1^N problem are as follows: $\text{Com}_{\text{AM}} = ND$ (Line 4), $\text{Com}_{\text{BM}} = (N + 1)D$ (Line 5 and Line 20). As for the communication between the servers, it is executed in the DVV sub-protocol. We have here $ND(D-1)$ due to the first part in the validation (Line 9) and $t(D - 1) < D(D - 1)$ due to the second part (Line 15). In addition, the servers have to communicate also in performing the scrambling (Line 8) in order to generate random shares of 0. That communication can take place offline and its cost is $ND(D-1)$. Hence, the overall communication cost among the servers is $\mathrm{Com}_{\mathrm{MM}} = (2N+1)D(D-1)$.

Before moving on, we consider Protocol 2* in which we replace Lines 6-12 of Protocol 2 with Sub-protocol 4. The communication cost of the original DVV in Protocol 2 was $2ND(D-1)$. However, the communication cost of Sub-protocol 4 is due to Lines 2 and 4 there and it is only $(\theta+1)D(D-1)$, where θ is the size of the random seeds σ_d, $d \in [D]$, in terms of field elements. (For example, we used a seed of 160 bits, so that $\theta = 160/\lceil \log p \rceil$.) Hence, the communication cost among the servers in Protocol 2* is only $\mathrm{Com}_{\mathrm{MM}} = (\theta+2)D(D-1)$, and that is a significant improvement in comparison to what we have in Protocol 2.

We move on to Protocol 3 for the OT_k^N problem. Its communication costs are as in Protocol 2 with one difference: at the end, Bob receives from each server N field elements and not just one. Hence, the costs of this protocol are:

$$\mathrm{Com}_{\mathrm{AM}} = ND, \quad \mathrm{Com}_{\mathrm{BM}} = 2ND, \quad \mathrm{Com}_{\mathrm{MM}} = (2N+1)D(D-1)\,. \tag{12}$$

The costs for Protocol 3*, that uses Sub-protocol 4 for DVV, are

$$\mathrm{Com}_{\mathrm{AM}} = ND, \quad \mathrm{Com}_{\mathrm{BM}} = 2ND, \quad \mathrm{Com}_{\mathrm{MM}} = (\theta+2)D(D-1)\,.$$

We note that the OT_k^N problem could also be solved by invoking Protocol 2 k times, where Alice's part has to be executed just once. The communication costs of this alternative course of action are:

$$\mathrm{Com}_{\mathrm{AM}} = ND, \quad \mathrm{Com}_{\mathrm{BM}} = k(N+1)D, \quad \mathrm{Com}_{\mathrm{MM}} = k(2N+1)D(D-1)\,. \tag{13}$$

Comparing Eq. (13) to Eq. (12) we see that such an alternative course of action is less efficient than Protocol 3 for every $k \geq 2$.

Next, we consider Protocol 5 for the problem of Priced OT. That protocol executes Protocol 3 (see Line 8 there), where part of the original DVV process in Protocol 3 is replaced with Sub-protocol 6. That modification leaves the communication costs of Protocol 3 unchanged. In addition, Protocol 5 includes Lines 1-7. Let us focus on the case where the weights are publicly known. Then Alice has to send D shares in T and so does Bob, so that adds D to both $\mathrm{Com}_{\mathrm{AM}}$ and $\mathrm{Com}_{\mathrm{BM}}$. The computation in Line 6 adds $t(D-1) < D(D-1)$ to $\mathrm{Com}_{\mathrm{MM}}$. Hence, we end up with the following costs:

$$\mathrm{Com}_{\mathrm{AM}} = (N+1)D, \quad \mathrm{Com}_{\mathrm{BM}} = (2N+1)D, \quad \mathrm{Com}_{\mathrm{MM}} = 2(N+1)D(D-1)\,.$$

The costs for Protocol 5*, that uses Sub-protocol 4 for DVV, are

$$\mathrm{Com}_{\mathrm{AM}} = (N+1)D, \quad \mathrm{Com}_{\mathrm{BM}} = (2N+1)D, \quad \mathrm{Com}_{\mathrm{MM}} = (\theta+3)D(D-1)\,.$$

Next, we consider the Generalized OT protocol, Protocol 7. The protocol performs OT_k^N (Protocol 3) twice, once for the vector of secrets (Line 4) and once for the vector of messages (Line 5); however, Bob's input vector is the same in the two OT_k^N executions, so the DVV sub-protocol is executed just once (Line 6). In addition, Alice distributes to the servers shares in s^A (Line 3), Bob does the same for s^B (Line 8), and the servers communicate in order to recover e (Line 10). Adding up everything yields the following costs:

$$\mathrm{Com}_{\mathrm{AM}} = (2N+1)D, \quad \mathrm{Com}_{\mathrm{BM}} = (3N+1)D, \quad \mathrm{Com}_{\mathrm{MM}} = (2N+2)D(D-1)\,.$$

The costs for Protocol 7*, that uses Sub-protocol 4 for DVV, are

$$\mathrm{Com}_{\mathrm{AM}} = (2N+1)D, \quad \mathrm{Com}_{\mathrm{BM}} = (3N+1)D, \quad \mathrm{Com}_{\mathrm{MM}} = (\theta+3)D(D-1)\,.$$

We proceed with Protocol 8 for the OMPE problem. The dimension of Alice's and Bob's vectors is $N_k := \binom{N+k}{k}$, and the number of entries in Bob's vector that need to be verified in the DVV sub-protocol is $N_k - k - 1$. In Lines 4+5 both Alice and Bob send to the servers $N_k D$ shares. Then, in the DVV sub-protocol, the servers send among themselves $D(D-1)$ field elements $N_k - k - 1$ times (for reconstructing ω in Line 9 for all relevant vector entries). Finally, the servers send to Bob D field elements (Line 14). The overall communication costs are therefore

$$\mathrm{Com}_{\mathrm{AM}} = N_k D, \quad \mathrm{Com}_{\mathrm{BM}} = (N_k+1)\,D, \quad \mathrm{Com}_{\mathrm{MM}} = (N_k-k-1)\,D(D-1)\,.$$

The communication costs of all protocols are summarized in Table 1.

7 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`.

Problem	Protocol	$\mathrm{Com}_{\mathrm{AM}}$	$\mathrm{Com}_{\mathrm{BM}}$	$\mathrm{Com}_{\mathrm{MM}}$
SP	1	$(N+1)D$	$(N+1)D$	0
OT_1^N	2	ND	$(N+1)D$	$(2N+1)D(D-1)$
	2*	ND	$(N+1)D$	$(\theta+2)D(D-1)$
OT_k^N	3	ND	$2ND$	$(2N+1)D(D-1)$
	3*	ND	$2ND$	$(\theta+2)D(D-1)$
Priced OT	5	$(N+1)D$	$(2N+1)D$	$2(N+1)D(D-1)$
	5*	$(N+1)D$	$(2N+1)D$	$(\theta+3)D(D-1)$
Generalized OT	7	$(2N+1)D$	$(3N+1)D$	$2(N+1)D(D-1)$
	7*	$(2N+1)D$	$(3N+1)D$	$(\theta+3)D(D-1)$
OMPE	8	N_kD	$(N_k+1)D$	$(N_k-k-1)D(D-1)$

Table 1: Communication costs of all distributed protocols with D servers. N denotes the dimension of the vectors in SP, the number of messages in all OT protocols, and the degree of the polynomials in the OMPE protocol. The parameter k in Protocol 8 denotes the number of variables, while $N_k = \binom{N+k}{k}$.

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

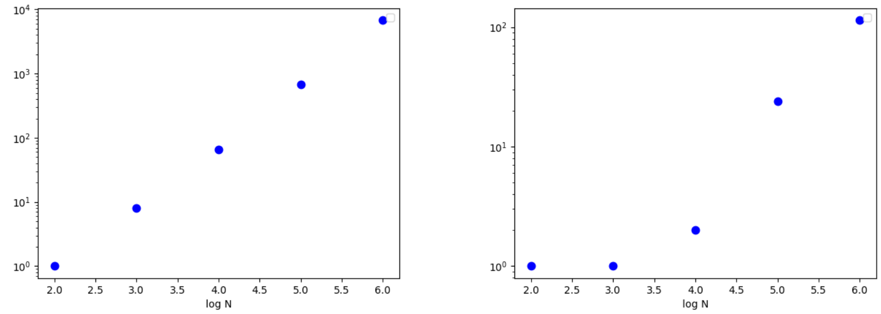


Figure 1: Runtimes (milliseconds) for Protocol 1 (DSP), as a function of $\log_{10}(N)$, for $D = 7$. The left plot shows the runtimes for Alice and Bob; the right plot shows the average runtimes for the servers. The runtimes are presented on a logarithmic scale.

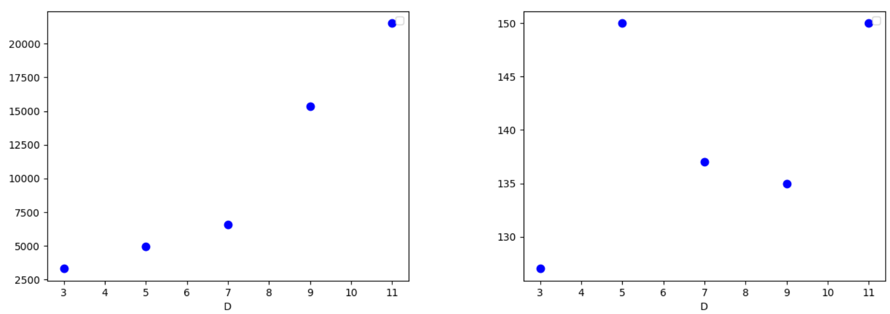


Figure 2: Runtimes (milliseconds) for Protocol 1 (DSP), as a function of D, for $N = 10^6$. The left plot shows the runtimes for Alice and Bob; the right plot shows the average runtimes for the servers. The runtimes are presented on a linear scale.

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.

In addition, we ran Protocol 3*, in which the DVV sub-protocol is executed by Sub-protocol 4. Here, the servers used a seed σ of 160 bits (see Lines 1-2 in Sub-protocol 4), and then, in order to generate a new random field

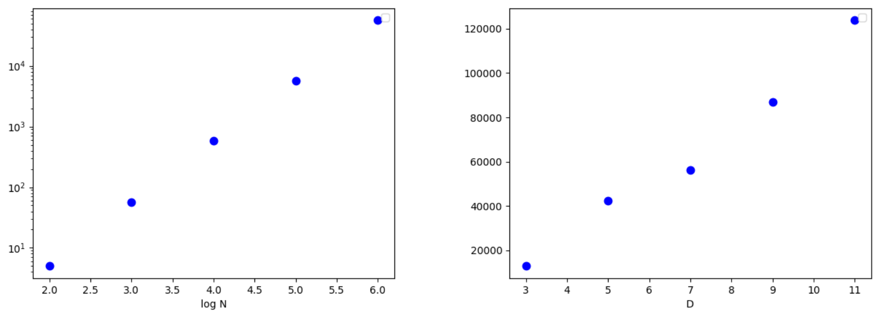


Figure 3: Average runtimes (milliseconds) for the servers in Protocol 3 (OT_k^N). Left: runtimes, on a logarithmic scale, as a function of $\log_{10}(N)$, for $D = 7$ and $k = 10$. Right: runtimes as a function of D, for $N = 10^6$ and $k = 10$.

element (as required in Line 3 in Sub-protocol 4) each server computes $\sigma \leftarrow$ SHA-1(σ) and takes σ mod p as the field element. The runtimes are given in Figure 4. As expected from our analysis in Section 6, the improvement in runtime is dramatic.

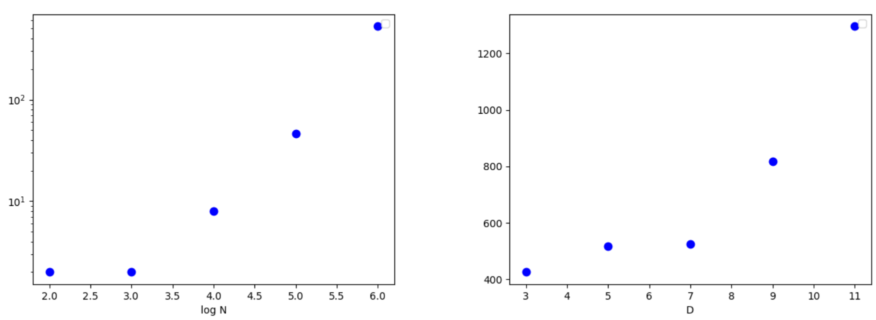


Figure 4: Average runtimes (milliseconds) for the servers in Protocol 3* (OT_k^N), where DVV is executed by Sub-protocol 4. Left: runtimes, on a logarithmic scale, as a function of $\log_{10}(N)$, for $D = 7$ and $k = 10$. Right: runtimes as a function of D, for $N = 10^6$ and $k = 10$.

We turn our attention to Protocol 5 (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

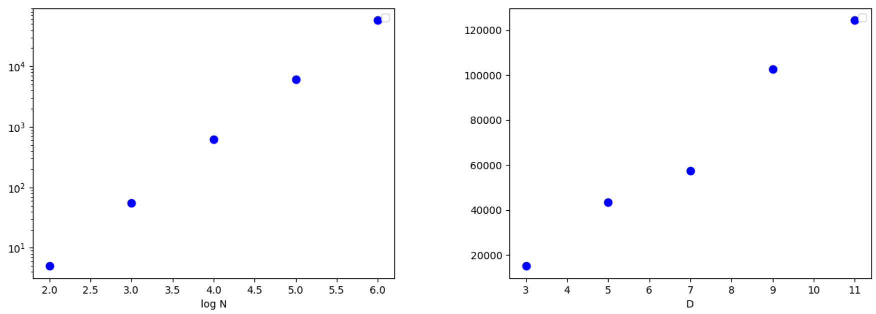


Figure 5: Average runtimes (milliseconds) for the servers in Protocol 5 (Priced OT)) . Left: runtimes as a function of N, for $T = 100$ and $D = 7$; the runtimes are presented on a logarithmic scale. Right: runtimes as a function of D, for $T = 100$ and $N = 10^6$.

5. Runtimes for Protocol 5* , in which the DVV sub-protocol is executed by Sub-protocol 4, are shown in Figure 6.

Next, we tested Protocol 7 (Generalized OT) with the access structure that we described in Section 4.3.1. In all of our experiments we used compartments of equal size, $|U_i| = 10$, $1 \leq i \leq r$. The runtimes for Bob and the servers, as a function of N and D, are reported in Figures 7 and 8. The average runtimes for the servers in Protocol 7*, where the DVV is executed by Sub-protocol 4, are shown in Figure 9. The comparison between those runtimes and those as reported in Figures 7 and 8 illustrate the advantages of Sub-protocol 4.

Finally, we consider Protocol 8 (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 10. The shown runtimes grow linearly with $\binom{N+k}{k}$, since that is the size of the two vectors in the scalar product.

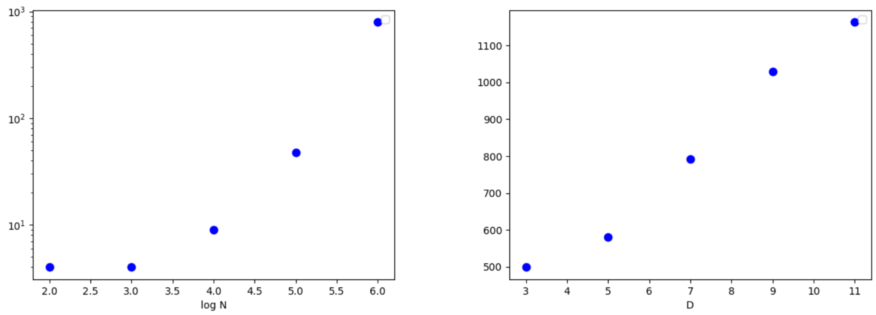


Figure 6: Average runtimes (milliseconds) for the servers in Protocol 5* (Priced OT)), where DVV is executed by Sub-protocol 4. Left: runtimes as a function of N, for $T = 100$ and $D = 7$; the runtimes are presented on a logarithmic scale. Right: runtimes as a function of D, for $T = 100$ and $N = 10^6$.

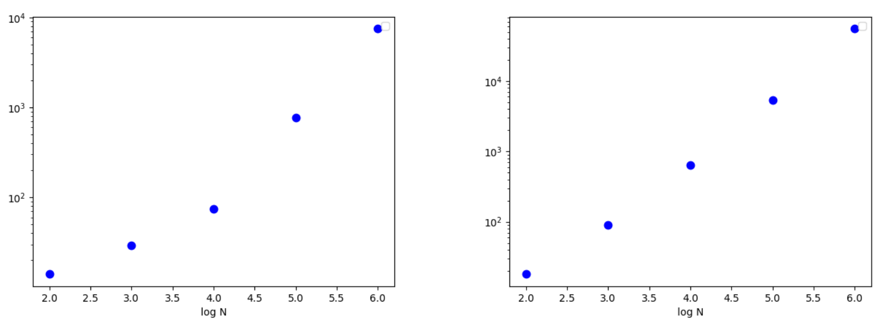


Figure 7: Runtimes (milliseconds) for Protocol 7 (GOT)), in the case of compartmented access structures as a function of N, for $D = 7$. The left plot shows the runtimes for Bob; the right plot shows the average runtimes for the servers. The runtimes are presented on a logarithm scale.

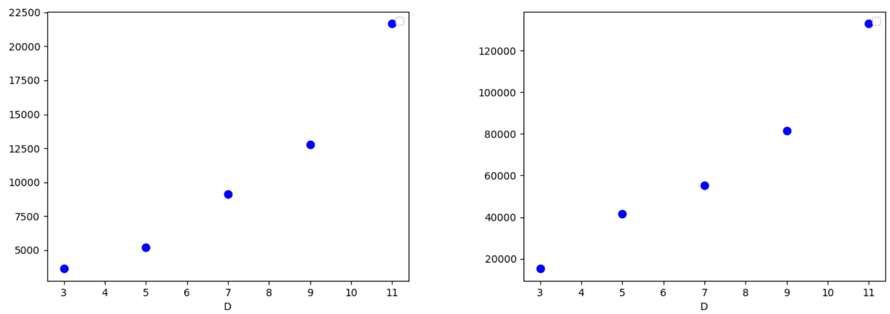


Figure 8: Runtimes (milliseconds) for Protocol 7 (GOT)), in the case of compartmented access structures as a function of D, for $N = 10^6$. The left plot shows the runtimes for Bob; the right plot shows the average runtimes for the servers. The runtimes are presented on a linear scale.

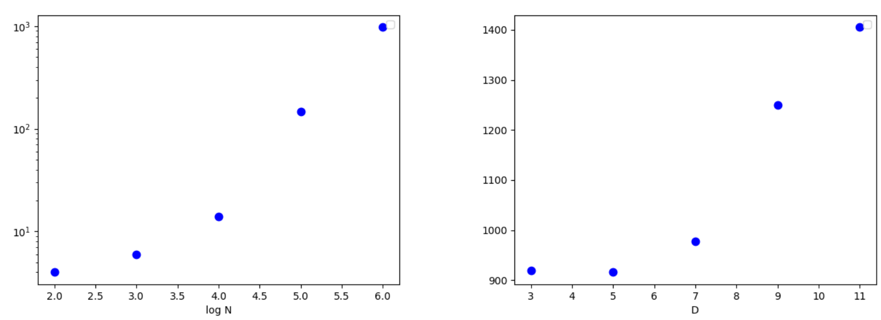
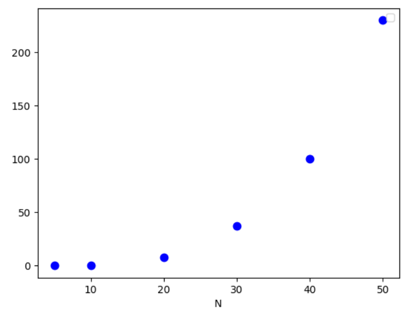


Figure 9: Runtimes (milliseconds) for Protocol 7* (GOT)), where DVV is executed by Sub-protocol 4, in the case of compartmented access structures. The left plot shows the average runtimes for the servers as a function of N, for $D = 7$, on a logarithmic scale; the right plot shows the average runtimes for the servers as a function of D, for $N = 10^6$ on a linear scale.



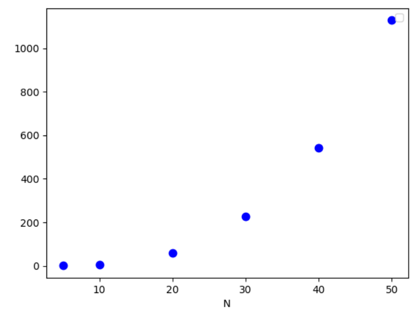


Figure 10: Runtimes (milliseconds) for Protocol 8 (OMPE), as a function of N, the polynomial degree, for $k = 3$. Left: runtimes for Bob; right: average runtimes for the servers.

8 The case of malicious servers

In our discussion so far we focused on the case of semi-honest servers. Here we describe cryptographic enhancements that render our protocols immune also when some of the servers are malicious. Those enhancements consist of modifications to the secret sharing sub-protocols only: dealing shares in a given secret, and the reconstruction of secrets from its shares; hence, they do not alter the information flow and logical structure of our distributed scalar product, OT and OPE protocols.

We demonstrate those enhancements only for the distributed protocol for the OT_1^N problem, as the modifications of other protocols go along the same lines. Moreover, we focus on Protocol 2*, which is obtained from Protocol 2 by replacing Lines 6-12 in it with Sub-protocol 4 (that verifies that Bob had submitted shares in a binary vector). We choose to focus on Protocol 2* as it is much more efficient than Protocol 2 (see Sections 6 and 7) and thus it is a more suitable starting point for security enhancements, as those impose a toll on communication and computation costs.

This section is structured as follows: in Section 8.1 we provide a quick overview of verifiable secret sharing; in Section 8.2 we describe the simple enhancements that would render our protocol secure when the number of malicious servers, denoted c, is smaller than $\frac{D}{4}$; finally, in Section 8.3 we describe the more involved enhancements that render our protocol secure when $c < \frac{D}{3}$.

8.1 A crash course on verifiable secret sharing

Verifiable secret sharing [12] is a protocol for sharing a secret in the presence of malicious adversaries. If the malicious adversary corrupts the dealer then the dealer may distribute to all parties ($M_1, \ldots, M_D$ in our case) shares that are not point values of some polynomial of degree $t-1$, where $t-1$ is the announced degree. If the malicious adversary corrupts some of the share-holding parties, they may submit in the reconstruction phase wrong share values, so that different subsets of t out of the D shares yield different secret values.

We begin by discussing the second problem, as it is easier to solve than the first one. Assume that the secret is s and that the dealer distributed to M_d, $d \in [D]$, the share $f(d)$, where $f(x) = \sum_{i=0}^{t-1} a_i x^i$ is a random polynomial of degree $t-1$ and $a_0 = s$. The collection of vectors

$$\mathcal{C} := \{(f(1), \ldots, f(D)) : (a_0, \ldots, a_{t-1}) \in \mathbb{Z}_p^t\}$$

is a $[D, t, D-t+1]$-linear code over $\mathbb{Z}_p$. That means that it is a linear subspace of $\mathbb{Z}_p^D$ of dimension t, and the Hamming distance between every two vectors in that subspace is at least $D-t+1$.[3] Such codes are called Reed-Solomon codes and there exists an efficient decoding algorithm that can correct in any given codeword $w \in \mathbb{Z}_p^D$ up to $(D-t)/2$ errors by looking for the codeword in $\mathcal{C}$ that minimizes the Hamming distance to w [41].

Hence, if in the reconstruction phase all D servers broadcast their shares, then assuming that at most $(D-t)/2$ among those shares are wrong, the servers can still reconstruct the correct codeword $(f(1), \ldots, f(D))$, and thus recover f, and consequently also $s = f(0)$.

As an example, if the number of corrupted parties c is less than $\frac{D}{3}$, the dealer can use a share-generating polynomial of degree $t-1$ where $t = \lceil \frac{D}{3} \rceil$. First, this setting of t ensures that the corrupted parties cannot recover the secret without at least one honest party, since if $c < \frac{D}{3}$ then $c \le t-1$. Moreover, as the Reed-Solomon decoding algorithm can correct up to $\frac{D-t}{2}$ errors, it can correct in this case all c wrong values that the corrupted parties may submit, since $\frac{D-t}{2} \ge c$. Indeed, $t = \lceil \frac{D}{3} \rceil$ satisfies $3t \le D+2$ and therefore $\frac{D-t}{2} \ge t-1 \ge c$.

The first problem, in which the dealer is malicious and distributes shares that are generated by a polynomial of degree greater than $t-1$, is somewhat

[3] Indeed, if f and g are two distinct polynomials of degree at most $t-1$, they can agree in at most $t-1$ points.

more involved. One way to prevent the dealer from cheating is described in [5, Section 5] and is outlined below.

The main idea is that the dealer uses a *bivariate* polynomial

$$S(x,y)=\sum_{i=0}^{t-1}\sum_{j=0}^{t-1}a_{i,j}x^iy^j$$

where $S(0,0)$ equals the underlying secret s. The dealer sends to each M_d, $d\in[D]$, two univariate polynomials — $f_d(x)=S(x,d)$ and $g_d(y)=S(d,y)$. It is easy to see that those univariate polynomials satisfy $f_d(d')=g_{d'}(d)$, for all $d,d'\in[D]$. Any two parties, M_d and $M_{d'}$, can then verify that

$$f_d(d')=g_{d'}(d) \quad\text{and}\quad f_{d'}(d)=g_d(d')\,. \tag{14}$$

If the equalities in Eq. (14) are verified by all $\binom{D}{2}$ pairs of parties, the parties are ascertained that the dealer had used a single bivariate polynomial $S(x,y)$ of degree $t-1$ [5, Claim 5.3]. In that case, the parties can recover the secret s since the values $f_d(0)=S(0,d)$, $d\in[D]$, induce a single univariate polynomial $S(0,y)$ of degree $t-1$, and $s=S(0,0)$. Furthermore, it can be shown that any $t-1$ parties learn no information on s (see [5, Claim 5.4]) so that this secret sharing scheme offers perfect secrecy.

We conclude by noting that the above described scheme, that uses a bivariate polynomial, requires private point-to-point channels between any pair of parties. (That assumption comes on top of the general assumption, as described in Section 2, that the servers have an authenticated broadcast channel.)

8.2 The case $c<\frac{D}{4}$

In Protocol 2*, the only parties that act as dealers of secrets are Alice and Bob (Lines 4 and 5). The servers validate the legality of Bob's input vector by reconstructing two secret values: one in Line 5 of Sub-protocol 4, and another in Line 15 of the main Protocol 2. Afterwards, Bob reconstructs the final output in Line 21.

First, we note that we should not worry about scenarios in which Alice or Bob deal their secrets in a malicious manner. As for Bob, being the receiver he is the only party that receives any output and, consequently, there is no reason for him to cheat. As for Alice, she might want to sabotage the process, but then she could simply submit a wrong input vector or refrain from participating in the protocol in the first place. Therefore, it makes no sense to verify that either Alice or Bob dealt their secrets properly.

Next, we discuss the necessary modifications to Protocol 2* in order to ascertain its correct and secure operation when $c < \frac{D}{4}$ of the servers are malicious. The modifications are:

1. In Lines 4-5 of the protocol, Alice and Bob use (t, D)-sharing with $t = \lceil D/4 \rceil$ instead of the original value of $t = \lfloor (D+1)/2 \rfloor$ (Eq. (1)).

2. When the servers compute ω in Line 15 they use all D shares. If the reconstruction procedure yields a polynomial of degree greater than $t-1$, the servers apply the Reed-Solomon decoding procedure to recover the correct value.

3. When the servers compute ω in Line 5 in Sub-protocol 4, if the D shares induce a polynomial of degree greater than $2t-2$, the servers apply the Reed-Solomon decoding procedure to recover the correct value.

4. When Bob computes the output in Line 21 in the main protocol, if the D shares induce a polynomial of degree greater than $2t-2$, Bob applies the Reed-Solomon decoding procedure to recover the correct value.

Theorem 3. *If $c < \frac{D}{4}$ then the above described modified Protocol 2* is correct and is perfectly secure.*

Proof. Assume that $D = 4j - i$ for $j \geq 1$ and $i \in \{0, 1, 2, 3\}$. Then $t = \lceil D/4 \rceil = j$ and $c \leq t - 1$. The degree of the secret sharing polynomials that Alice and Bob use is $t - 1$ and, therefore, at least t servers need to collaborate in order to recover any of the secrets. Since $c \leq t - 1$, the malicious servers cannot learn any information on the shared secrets.

When the servers compute ω in Line 15, the secret sharing polynomial is of degree $t - 1$. In that case, the underlying Reed-Solomon code can correct up to $(D - t)/2$ errors. But $\frac{D-t}{2} = \frac{3j-i}{2} \geq j - 1$, where the last inequality is equivalent to $j \geq i - 2$, which obviously holds for $j \geq 1$ and $i \in \{0, 1, 2, 3\}$. Since $j - 1 \geq c$, it follows that the code can correct all (up to) c errors.

In the other two places where a secret is interpolated (Line 5 in Sub-protocol 4 and Line 21 in the main protocol) the underlying polynomial is of degree $2t - 2$. In that case, the underlying Reed-Solomon code can correct up to $(D-(2t-1))/2$ errors. In this case we have

$$\frac{D-(2t-1)}{2} = \frac{4j-i-2j+1}{2} \geq j-1\,,$$

where the last inequality follows from the fact that $i \leq 3$. Since $j-1 \geq c$, here too it is possible to correct all (up to) c errors. □

We note that the communication and computational costs of the above described modified protocol are almost the same as those of Protocol 2*. Indeed, if there are no malicious servers then no Reed-Solomon decoding needs to be executed and the only difference is in Line 15 of the main protocol, where this time the servers need to use all D shares for interpolation and not just $\lfloor (D+1)/2 \rfloor$ of the shares as done in the original Protocol 2*. In case there are malicious servers, then the servers and Bob would need to run the Reed-Solomon decoding algorithm up to three times, where the complexity of this operation is independent of N, hence, the added cost is negligible.

8.3 The case $c < \frac{D}{3}$

So far we were able to dodge the need to run the BGW protocol [7] since all of the arithmetic expressions that we compute are polynomials of degree at most 2 in the shared secrets. Hence, by using secret sharing polynomials of degree at most $\lfloor (D+1)/2 \rfloor - 1$, it is possible to compute polynomials of degree at most 2 (namely, polynomials in which each term has no more than a single multiplication of secrets), without performing degree reduction, which is the essence and the most costly part of the BGW protocol.

However, if we aim to strengthen our protocols so that they would be immune against higher numbers of malicious servers, we would need to apply the BGW protocol, which involves degree reduction. That part of the computation requires each server to act also as a dealer of secret shares. Malicious servers may try to seize this opportunity in order to distribute shares that do not correspond to a share-generating polynomial of the proper degree. Hence, it would be necessary to implement techniques, such as the one described in Section 8.1, to prevent such malicious conduct.

Therefore, one can enhance the protocol by implementing the BGW protocol with verifiable secret sharing that offers security in the presence of a malicious adversary who controls $c < \frac{D}{3}$ servers. That protocol, as described in [7], was shown in [5, Section 6] to be perfectly secure when $c < D/3$. The interested reader is referred to [5, 7] for a detailed description and analysis of that protocol.

We proceed to summarize the needed modifications in Protocol 2* that would render it secure against an adversary that corrupts $c < \frac{D}{3}$ of the servers:

1. In Lines 4-5 of the protocol, Alice and Bob use (t, D)-sharing with $t = \lceil D/3 \rceil$.

2. When the servers compute ω in Line 15 they use all D shares. If the reconstruction procedure yields a polynomial of degree greater than $t-1$, the servers apply the Reed-Solomon decoding procedure to recover the correct value.

3. The computation in Line 4 in Sub-protocol 4 is carried out by applying the BGW multiplication procedure in the presence of adversaries.

4. When computing ω in Line 5 of Sub-protocol 4, if the D shares induce a polynomial of degree greater than $t - 1$, the servers apply the Reed-Solomon decoding procedure to recover the correct value.

5. The computation in Line 19 in the main protocol is carried out by applying the BGW multiplication procedure in the presence of adversaries.

6. When Bob computes the output in Line 21 in the main protocol, if the D shares induce a polynomial of degree greater than $t - 1$, Bob applies the Reed-Solomon decoding procedure to recover the correct value.

As noted earlier, the costly part in those enhancements is the multiplication procedure. If the servers generate offline (before Bob submits his input) a sufficient number of Beaver triplets [6], those triplets can be later used in order to significantly reduce response time to queries that receivers submit.

9 Related work

Naor and Pinkas [30] 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 distributed OT_1^2, meaning that Bob has to communicate with ℓ out of the D servers in order to receive his message of choice.[4].

The two protocols that are proposed in [30] 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_1m_1, and c_2m_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. [8] generalized the protocols of [30] 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 [30], Bob contacts

[4]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.

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 [30]: 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 [30] and [8] 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 distributed OT protocol.

Nikov et al. [31] presented an analysis of the ℓ-out-of-D distributed OT_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 [31, 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 distributed OT_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 [8, 31] 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 [15] 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 distributed OT. Their protocol is similar to our distributed OT_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

[15] 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 [15] 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 [15], 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 [29]. 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 [29] 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. [28]. 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 [14] described another DOPE protocol that offers better security and complexity than the protocols of Li et al. [28]. 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. If $f(x) = \sum_{n=0}^{N} a_n x^n$, Alice generates random values $r_1, \ldots, r_N$ and then distributes to the servers shares in a_n, $0 \leq n \leq N$, and in $\gamma_n := r_n a_n$, $1 \leq n \leq N$, where the secret sharing polynomials are of degree $t-1$. In addition, she sends $r_1, \ldots, r_N$ to Bob. Subsequently, Bob broadcasts to all servers the values $e_n := \alpha^n - r_n$, $1 \leq n \leq N$. Then, a subset of t servers, say $M_1, \ldots, M_t$, respond to Bob as follows: server M_d, $1 \leq d \leq t$, sends to Bob the value $[z]_d := [a_0]_d + \sum_{j=1}^{N} (e_n[a_n]_d + [\gamma_n]_d)$, where $[x]_d$ denotes M_d's share in the value x. Those values enable Bob to reconstruct a polynomial $Z(x)$ of degree t such that $Z(0) = f(\alpha)$. Despite the advantages that their protocol offers with respect to that of Li et al. [28], 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 8 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 multivariate polynomials.

We note that all of the related work discussed above assumes a passive adversarial model. Namely, an adversary may corrupt some of the servers in order to get hold of their view in the protocol, but it cannot actively change the messages that they send. Namely, those protocols provide security against a collusion between some of the servers, assuming that they are still semi-honest, but not against malicious servers who may act arbitrarily. In our discussion in Section 8 we explained how to enhance our protocols so that they provide protection even when some of the servers are malicious.

10 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 allows. 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. Finally, ours are the first distributed MPC protocols that are designed also for the case of malicious servers. Our protocols are information-theoretic secure and provide unconditional security to both Alice and Bob, even when some of the parties collude, and they are very efficient.

The protocols that we presented herein 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.

While OT and OPE can serve as building blocks for general MPC problems [25, 28], 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. We note that Boolean circuits can be represented as multivariate polynomials over $\mathbb{Z}_2$ in the k binary input wires; hence, that problem is a special case of OMPE. However, as it can be impractical to represent a Boolean circuit by a polynomial, another approach that evaluates the circuit gate by gate seems more suitable.

• 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] Benny Applebaum, Amos Beimel, Oded Nir, and Naty Peter. Better secret sharing via robust conditional disclosure of secrets. In *STOC*, pages 280–293, 2020.

[5] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *J. Cryptol.*, 30:58–151, 2017.

[6] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, pages 420–432, 1991.

[7] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[8] 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.

[9] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *CRYPTO*, pages 234–238, 1986.

[10] Ernest F. Brickell. Some ideal secret sharing schemes. In *EUROCRYPT*, pages 468–475, 1989.

[11] Octavian Catrina and Florian Kerschbaum. Fostering the uptake of secure multiparty computation in e-commerce. In *ARES*, pages 693–700, 2008.

[12] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *FOCS*, pages 383–395.

[13] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[14] Louis Cianciullo and Hossein Ghodosi. Unconditionally secure oblivious polynomial evaluation: A survey and new results. *J. Comput. Sci. Technol.*, 37(2):443–458, 2022.

[15] Christian L. F. Corniaux and Hossein Ghodosi. Scalar product-based distributed oblivious transfer. In *ICISC*, pages 338–354, 2010.

[16] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005.

[17] 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.

[18] Lihi Dery, Tamir Tassa, Avishay. Yanai, and Arthur Zamarin. Demo: A secure voting system for score based elections. In *CCS*, pages 2399–2401, 2021.

[19] Wenliang Du and Mikhail J. Atallah. Privacy-preserving cooperative statistical analysis. In *ACSAC*, pages 102–110, 2001.

[20] Wenliang Du and Justin Zhijun Zhan. A practical approach to solve secure multi-party computation problems. In *NSPW*, pages 127–135, 2002.

[21] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[22] Alon Ben Horin and Tamir Tassa. Privacy preserving collaborative filtering by distributed mediation. In *RecSys*, pages 332–341, 2021.

[23] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *ISTCS*, pages 174–184, 1997.

[24] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, 2011. 272.

[25] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[26] 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.

[27] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *FOCS*, pages 364–373, 1997.

[28] 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.

[29] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *STOC*, pages 245–254, 1999.

[30] Moni Naor and Benny Pinkas. Distributed oblivious transfer. In *ASIACRYPT*, pages 205–219, 2000.

[31] Ventzislav Nikov, Svetla Nikova, Bart Preneel, and Joos Vandewalle. On unconditionally secure distributed oblivious transfer. In *INDOCRYPT*, pages 395–408, 2002.

[32] Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[33] Johannes Schneider. Lean and fast secure multi-party computation: Minimizing communication and local computation using a helper. In *SECRYPT*, pages 223–230, 2016.

[34] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[35] Erez Shmueli and Tamir Tassa. Mediated secure multi-party protocols for collaborative filtering. *ACM Transactions on Intelligent Systems and Technology*, 11:1–25, 2020.

[36] Tamir Tassa. Generalized oblivious transfer by secret sharing. *Des. Codes Cryptogr.*, 58(1):11–21, 2011.

[37] Tamir Tassa and Nira Dyn. Multipartite secret sharing by bivariate interpolation. In *ICALP*, pages 288–299, 2006.

[38] Tamir Tassa, Tal Grinshpoun, and Avishay Yanai. PC-SyncBB: A privacy preserving collusion secure DCOP algorithm. *Artificial Intelligence*, 297:103501, 2021.

[39] Tamir Tassa, Ayman Jarrous, and Yonatan Ben-Ya'akov. Oblivious evaluation of multivariate polynomials. *J. Math. Cryptol.*, 7(1):1–29, 2013.

[40] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. In *SIGKDD*, pages 639–644, 2002.

[41] Stephen B. Wicker and Vijay K. Bhargava. *Reed-Solomon Codes and their Applications.* John Wiley Sons, 1999.

[42] Andrew C. Yao. Protocols for secure computation. In *FOCS*, pages 160–164, 1982.

A A 1-out-of-N distributed OT protocol with non-interacting servers

The central part in Protocol 2 was the DVV sub-protocol, namely, the part in which the servers validate that Bob had submitted a legal vector $\mathbf{b} \in W = \{\mathbf{e}_j : j \in [N]\}$. That part is essential in order to verify that Bob learns exactly one of the secret messages and not an arbitrary linear combination of them. This is the only part in the protocol where the servers communicate with each other.

The alternative protocol achieves the same goal as the DVV sub-protocol by applying a different mechanism. In that protocol Alice generates a random vector of nonzero multipliers $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N) \in (\mathbb{Z}_p^*)^N$ (where $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$). Then, she performs her part in Protocol 2 (Lines 1 and 3-4) twice: once with the vector of masked messages $\mathbf{a}_{\boldsymbol{\alpha}} = (\alpha_1 m_1, \ldots, \alpha_N m_N)$, and once with the vector of random multipliers $\boldsymbol{\alpha}$. Bob performs his part in the initial stage of Protocol 2 (Lines 2, 3+5) exactly the same. After completing this initial stage, the servers skip to the last stage in Protocol 2 (Lines 18-21). But now that part is performed twice so that Bob recovers two scalar products — $\mathbf{a}_{\boldsymbol{\alpha}} \cdot \mathbf{b}$ and $\boldsymbol{\alpha} \cdot \mathbf{b}$. If Bob had submitted a legal selection vector $\mathbf{b} = \mathbf{e}_j$, $j \in [N]$, then $\mathbf{a}_{\boldsymbol{\alpha}} \cdot \mathbf{b} = \alpha_j m_j$ and $\boldsymbol{\alpha} \cdot \mathbf{b} = \alpha_j$. Since $\alpha_j \neq 0$ Bob can recover the message of his choice, m_j. However, if Bob had submitted $\mathbf{b} \notin W = \{\mathbf{e}_j : j \in [N]\}$, he will get no information at all.

Theorem 4. *If Bob submits an illegal vector* $\mathbf{b} \notin W = \{\mathbf{e}_j : j \in [N]\}$*, the two scalar products* $\mathbf{a}_{\boldsymbol{\alpha}} \cdot \mathbf{b}$ *and* $\boldsymbol{\alpha} \cdot \mathbf{b}$ *reveal no information on the messages* $m_1, \ldots, m_N$.

The proof of Theorem 4 is given below. We note that the mechanism that we used here to ensure that Bob respects the protocol is based on an idea that was presented in [30] for their 1-out-of-2 OT protocol. This approach is advantageous with respect to our DVV mechanism as it does not require the servers to communicate with each other. However, it enforces Alice to be online whenever a client (Bob) wishes to engage in an information retrieval since she needs to generate a new set of random masks for each request. So, for example, if Alice holds some database that serves multitude of clients ("Bob"s), while our approach enables Alice to act only once and then, whenever a client wishes to retrieve a record from the database he only needs to communicate with the servers, in the approach presented here Alice (as well as the servers) have to be responsive to each client. Another problem with the approach presented here is that it is tailored to 1-out-of-N OT. We solve herein many other problems of OT and OPE: while the approach that we used here for 1-out-of-N OT does not extend to such problems, the DVV approach does, as we show hereinafter.

Proof of Theorem 4. Denote by q the number of entries in $\mathbf{b}$ that are nonzero. Let us assume, without loss of generality, that $b_n \neq 0$ for all $1 \leq n \leq q$ and $b_n = 0$ for all $q < n \leq N$. Then Bob will receive two values from the two stages in the distributed OT protocol — $\sum_{n=1}^{q} b_n \alpha_n m_n$ and $\sum_{n=1}^{q} b_n \alpha_n$. Since α_n distribute uniformly in $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ and $b_n \neq 0$, $1 \leq n \leq q$, those two values are of the form

$$U := \sum_{n=1}^{q} \beta_n m_n \quad \text{and} \quad V := \sum_{n=1}^{q} \beta_n \,, \tag{15}$$

where β_n distribute uniformly in $\mathbb{Z}_p^*$.

Our goal is now to show that U and V reveal no information on any given linear combination of $(m_1, \ldots, m_q)$. Namely, that if $(c_1, \ldots, c_q)$ is any arbitrary nonzero vector in $\mathbb{Z}_p^q$ and $X = \sum_{n=1}^{q} c_n m_n$, then U and V reveal no information on X. To do that we view U, V and X as random variables in $\mathbb{Z}_p$. Their value is determined by the random vectors $(\beta_1, \ldots, \beta_q)$ and $(m_1, \ldots, m_q)$. The first distributes uniformly in $(\mathbb{Z}_p^*)^q$, while the second distributes uniformly in $\mathbb{Z}_p^q$. Our goal is to show that $H(X|U, V) = H(X)$, namely, that the conditional entropy of X, given U and V, equals the a-priori entropy of X. We show that by proving that for any three scalars u, v, x we have

$$P(X = x | U = u, V = v) = P(X = x) \,. \tag{16}$$

We begin by the considering the case $(c_1, \ldots, c_q) = (1, 0, \ldots, 0)$. In this case $X = m_1$. Hence, as we assume that Bob has no prior knowledge on the value of the messages $m_1, \ldots, m_N$, the probability on the right hand side of Eq. (16) equals

$$P(X = x) = \frac{1}{p} \,, \quad \forall x \in \mathbb{Z}_p \,. \tag{17}$$

The conditional probability on the left hand side of Eq. (16) is

$$P\left(m_1 = x \;\middle|\; \sum_{n=1}^{q} \beta_n m_n = u \text{ and } \sum_{n=1}^{q} \beta_n = v \right) . \tag{18}$$

Let us fix a specific tuple $(\beta'_1, \ldots, \beta'_q)$ such that $\sum_{n=1}^{q} \beta'_n = v$. Then

$$P\left(m_1 = x \;\Big|\; \sum_{n=1}^{q} \beta'_n m_n = u\right) = \frac{P\left(m_1 = x \text{ and } \sum_{n=1}^{q} \beta'_n m_n = u\right)}{P\left(\sum_{n=1}^{q} \beta'_n m_n = u\right)} . \quad (19)$$

As the messages $m_1, \ldots, m_N$ are independent, the probability in the numerator on the right hand side of Eq. (19) equals

$$P\left(m_1 = x \text{ and } \sum_{n=1}^{q} \beta'_n m_n = u\right) = P(m_1 = x) \cdot P\left(\sum_{n=2}^{q} \beta'_n m_n = u - \beta'_1 x\right) . \quad (20)$$

Since all messages distribute uniformly in $\mathbb{Z}_p$ we infer that each of the two multiplicands on the right hand side of Eq. (20) equals $\frac{1}{p}$. For the same reason also the probability in the denominator on the right hand side of Eq. (19) equals $\frac{1}{p}$. It follows that

$$P\left(m_1 = x \;\Big|\; \sum_{n=1}^{q} \beta'_n m_n = u\right) = \frac{p^{-1} \cdot p^{-1}}{p^{-1}} = \frac{1}{p} . \quad (21)$$

Since $(\beta'_1, \ldots, \beta'_q)$ is an arbitrary tuple that satisfies $\sum_{n=1}^{q} \beta'_n = v$, Eqs. (18)+(21) imply that

$$P(X = x | U = u, V = v) = \frac{1}{p} . \quad (22)$$

Hence, Eq. (16) follows from Eqs. (17)+(22).

The general case where $(c_1, \ldots, c_q)$ is any nonzero vector in $\mathbb{Z}_p^q$ can be reduced to the case $(c_1, \ldots, c_q) = (1, 0, \ldots, 0)$ by applying a suitable linear transformation. As $(c_1, \ldots, c_q) \neq \mathbf{0}$ we can assume, without loss of generality, that $c_1 = 1$. Define $m'_1 = \sum_{n=1}^{q} c_n m_n$, and $m'_n = m_n$ for all $2 \leq n \leq q$. Then the conditional probability on the lefy of Eq. (16) equals

$$P\left(m'_1 = x \;\Big|\; \sum_{n=1}^{q} \beta'_n m'_n = u, \text{ and } \sum_{n=1}^{q} \beta_n = v\right) , \quad (23)$$

where $\beta'_n = \beta_n - \beta_1 c_n$, $1 \leq n \leq q$. Since also m'_n, $1 \leq n \leq q$, distribute independently and uniformly on $\mathbb{Z}_p$, arguing along the same lines as above shows that the probability in Eq. (23) equals $\frac{1}{p}$, and also $P(m'_1 = x) = \frac{1}{p}$. That settles our claim for any arbitrary linear combination of $m_1, \ldots, m_N$ in which there are $q > 1$ nonzero coefficients. $\square$

B Proof of Lemma 1

Assume that $\mathbf{b}$ is as in Eq. (11). Then for every multi-index $\mathbf{j} \in \mathbb{Z}_+^{k,N}$, the corresponding entry in $\mathbf{b}$ is $b_{\mathbf{j}} := \boldsymbol{\alpha}^{\mathbf{j}}$. Hence, for any $2 \le n \le N$ and for any $\mathbf{j} \in T_n$, there exists at least one monomial $\mathbf{h} \in T_{n-1}$ such that $\mathbf{j} = \mathbf{h} + \mathbf{e}_i$, for some $1 \le i \le k$. Let us compare the monomial $b_{\mathbf{j}} := \boldsymbol{\alpha}^{\mathbf{j}}$ with the monomial $b_{\mathbf{h}} := \boldsymbol{\alpha}^{\mathbf{h}}$. The two multi-indices $\mathbf{j}$ and $\mathbf{h}$ equal in all entries except for the ith entry, where $j_i = h_i + 1$. Therefore,

$$b_{\mathbf{j}} := \boldsymbol{\alpha}^{\mathbf{j}} = \boldsymbol{\alpha}^{\mathbf{h}} \cdot \alpha_i = b_{\mathbf{h}} \cdot b_{\mathbf{e}_i} \, .$$

Hence, such a vector will pass all stages of the DVV in Lines 6-11.

Assume next that $\mathbf{b}$ does not comply with the form as in Eq. (11). That means that

$$\mathbf{b} = (1, \alpha_1, \ldots, \alpha_k, b_{\mathbf{j}} : 2 \le |\mathbf{j}| \le N) \, ,$$

where there exists at least one entry $b_{\mathbf{j}}$, where $2 \le |\mathbf{j}| \le N$, that is not of the form as in Eq. (11). Let us focus on the first multi-index $\mathbf{j}$ that is not of that form. Namely, $\mathbf{j}$ is the first multi-index for which

$$b_{\mathbf{j}} \ne \boldsymbol{\alpha}^{\mathbf{j}} \, , \tag{24}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$. Assume that $|\mathbf{j}| = n \in [2, N]$ and let i be any index between 1 and k such that $\mathbf{j} = \mathbf{h} + \mathbf{e}_i$ for some $\mathbf{h} \in T_{n-1}$. By the minimality of $\mathbf{j}$ it means that

$$b_{\mathbf{h}} = \boldsymbol{\alpha}^{\mathbf{h}} \, . \tag{25}$$

From Eqs. (24) and (25) it follows that the validation check in Lines 9+10 would fail for those multi-indices. That completes the proof. $\Box$