Two-Round Oblivious Linear Evaluation from Learning with Errors

Pedro Branco[1], Nico Döttling[2], and Paulo Mateus[1]

[1]IT, IST - University of Lisbon
[2]Helmholtz Center for Information Security (CISPA)

Abstract

Oblivious Linear Evaluation (OLE) is a simple yet powerful cryptographic primitive which allows a sender, holding an affine function $f(x) = a + bx$ over a finite field, to let a receiver learn $f(w)$ for a w of the receiver's choice. In terms of security, the sender remains oblivious of the receiver's input w, whereas the receiver learns nothing beyond $f(w)$ about f. In recent years, OLE has emerged as an essential building block to construct efficient, reusable and maliciously-secure two-party computation.

In this work, we present efficient two-round protocols for OLE based on the Learning with Errors (LWE) assumption. Our first protocol for OLE is secure against malicious unbounded receivers and semi-honest senders. The receiver's first message may carry information about a batch of inputs, and not just a single input. We then show how we can extend the above protocol to provide malicious security for both parties.

1 Introduction

Oblivious Linear Evaluation (OLE) is a cryptographic primitive between a sender and a receiver, where the sender inputs an affine function $f(x) = a + bx$ over a finite field $\mathbb{F}$, the receiver inputs an element $w \in \mathbb{F}$, and in the end the receiver learns $f(w)$. The sender remains oblivious of the receiver's input w and the receiver learns nothing beyond $f(w)$ about f. OLE can be seen as a generalization of the well-known Oblivious Transfer (OT) primitive.[1] In fact, just as secure computation of *Boolean* circuits can be based on OT, secure computation of *arithmetic* circuits can be based on OLE [GMW87, IPS09].

In recent years, OLE has emerged as one of the most promising avenues to realize efficient two-party secure computation in different settings [IPS09, ADI$^+$17, DGN$^+$17, BCGI18, HIMV19, CDI$^+$19]. Interestingly, OLE has found

[1]It is easy to see that, if we consider the affine function $f : \{0, 1\} \to \{0, 1\}$ such that $f(b) = m_0 + b(m_1 - m_0)$, OLE trivially implements OT.

applications, not just in the secure computation of generic functions, but also in specific tasks such as Private Set Intersection [GN19, GS19] or Machine Learning related tasks [MZ17, JVC18].

Other aspects that set OLE apart from OT are reusability, meaning that the first message of a protocol is reusable across multiple executions,[2] and the fact that even a semi-honest secure OLE can be used to realize maliciously secure two-party computation [HIMV19].

Although OLE secure against semi-honest adversaries is complete for maliciously-secure two-party computation [HIMV19], this comes at the cost of efficiency and, thus, is it always preferable to start with a maliciously-secure one. Moreover, some applications of OLE even ask specifically for a maliciously-secure one [GN19]. Given this state of affairs and the importance of OLE in constructing two-party secure computation protocols, we ask the following question:

Can we build efficient and maliciously-secure two-round OLE protocols from (presumed) post-quantum hardness assumptions?

1.1 Our Results

In this work, we give an affirmative answer to the question above. Specifically, we present two simple, efficient and round-optimal protocols for OLE based on the hardness of the Learning with Errors (LWE) assumption [Reg05], which is conjectured to be post-quantum secure.

Before we start, we clarify what type of OLE we obtain. OLE comes in many flavors, one of the most useful being *vector OLE* where the sender inputs two vectors $a = \mathbf{a}, b = \mathbf{b} \in \mathbb{F}^\ell$ and the receiver obtains a linear combination of them $\mathbf{z} = \mathbf{a} + w\mathbf{b} \in \mathbb{F}^\ell$ [BCGI18]. For simplicity, we just refer to this variant as OLE.

Both of our protocols implement the functionality in just two-rounds and have the following properties:

- Our first protocol (Section 5) for OLE achieves statistical security against a corrupted receiver and computational semi-honest security against a corrupted sender. Additionally, we show how we can extend this protocol to implement *batch OLE*, a functionality similar to OLE where the receiver can input a batch of values $\{x_i\}_{i\in[k']}$, instead of just one value.

- We then show how to extend the above protocol to provide malicious security for both parties (Section 6). The protocol makes λ invocations of a two-round Oblivious Transfer protocol (which exists under LWE [PVW08, DGH$^+$20]), where λ is the security parameter. By instantiating the OT

[2]While two-party *reusable* non-interactive secure computation (NISC) is impossible in the OT-hybrid model [CDI$^+$19], reusable NISC for general Boolean circuits is known to be possible in the (reusable) OLE-hybrid model assuming one-way functions [CDI$^+$19]. The result stated above is meaningful only if we have access to a reusable two-round OLE protocol. The only efficient realizations of this primitive are based on the Decisional Composite Residuosity (DCR) and the Quadratic Residuosity assumptions [CDI$^+$19].

with the LWE-based protocols of [PVW08, Qua20], we preserve statistical security against a malicious receiver.

1.2 Related Work and Comparison

In the following, we briefly review some proposals from prior work and compare them with our proposal. We only consider schemes that are provable UC-secure as our protocols. OLE can be trivially implemented using Fully/Somewhat Homomorphic Encryption (e.g., [JVC18]) but these solutions are usually just proven secure against semi-honest adversaries and it is unclear how to extend security against malicious adversaries without relying on generic approaches such as Non-Interactive Zero-Knowledge (NIZK) proofs.[3] OLE can also be trivially implemented using generic solutions for two-party secure computation (via OT) such as [GS18, BL18]. However, these solutions fall short in achieving an *acceptable* level of efficiency.

The work of Döttling et al. [DKM12, DKMQ12] proposed an OLE protocol with unconditional security, in the stateful tamper-proof hardware model. The protocol takes only two rounds, however further interaction with the token is needed by the parties.

In [IPS09], a semi-honest protocol for oblivious multiplication was proposed, which can be easily extended to a OLE protocol. The protocol is based on noisy encodings. Based on the same assumption, [GNN17] proposed a maliciously-secure OLE protocol, which extends the techniques of [IPS09]. However, their protocol takes eight rounds of interaction.

Chase et al. [CDI$^+$19] presented a round-optimal reusable OLE protocol based on the Decisional Composite Residuosity (DCR) and the Quadratic Residuosity (QR) assumptions. The protocol is maliciously-secure and, to the best of our knowledge, it is the most efficient protocol for OLE proposed so far. However, it is well-known that both the DCR and the QR problems are quantumly insecure.

We also remark that our protocols implement vector OLE where the sender's input are vectors over a field, as in [GNN17].

In Table 1, a brief comparison between several UC-secure OLE protocols is presented.

1.3 Open Problems

Our first protocol is secure against semi-honest senders and, thus, it is trivially reusable. However, our fully maliciously-secure protocol (in Section 6) does not have reusability of the first message. Hence, the main open problem left in our work is the following: Can we construct a reusable maliciously-secure two-round OLE protocol based on the LWE assumption?

[3]As an example consider the work of [CDI$^+$19], where the Paillier cryptosystem is extended into an OLE protocol with malicious security and the construction is highly non-trivial.

	Hardness Assumption	Setup Assumption	Rounds	Reusability	Security
[IPS09]	Noisy Encodings	OT	3	-	semi-honest
[DKM12]	-	Stateful tamper proof hardware	2	-	malicious
[GNN17]	Noisy Encodings	OT	8	-	malicious
[CDI$^+$19]	DCR & QR	CRS	2	✓	malicious
This work	LWE	CRS	2	✓	malicious receiver
	LWE	CRS & OT	2	✗	malicious

Table 1: Comparison between different OLE schemes.

2 Technical Outline

In this section, we give a brief overview of our protocols.

2.1 A Two-Round Semi-Honest Protocol

In our protocol, both the sender S and the receiver R have access to a common reference string $\mathsf{crs} = \mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{k\times n}$. The core idea of our protocol is the following:

1. R computes $\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e}$ where $\mathbf{e}$ is a short noise vector and the i-th coordinate $x_i \in \mathbb{Z}_q$ of $\mathbf{x}$ corresponds to its input.
2. S samples a *short* matrix $\mathbf{R}$ and computes the pair $(\mathbf{s}_0 = \mathbf{a}'\mathbf{R}, \mathbf{s}_1 = \mathbf{a}_i\mathbf{R})$ where $\mathbf{a}_i$ is the i-th row of $\mathbf{A}$. It sends $\mathbf{A}_{-i}\mathbf{R}$ to the receiver, where $\mathbf{A}_{-i}$ is the matrix $\mathbf{A}$ with the i-th row removed.
3. R computes $\tilde{\mathbf{y}} = \mathbf{x}_{-i}\mathbf{C}$ where $\mathbf{x}_{-i}$ is the vector $\mathbf{x}$ with the i-th coordinate removed.

Observe that

$$\begin{aligned}\tilde{\mathbf{y}} &= \mathbf{x}_{-i}\mathbf{C} = \mathbf{x}_{-i}\mathbf{A}_{-i}\mathbf{R} = (\mathbf{x}\mathbf{A} - x_i\mathbf{a}_i)\mathbf{R}\\ &= (\mathbf{x}\mathbf{A} + \mathbf{e})\mathbf{R} - (x_i\mathbf{a}_i)\mathbf{R} - \mathbf{e}\mathbf{R} = \mathbf{s}_0 + x_i\mathbf{s}_1 + \mathbf{e}'\end{aligned}$$

for some short vector $\mathbf{e}'$. Hence, the vector obtained by R is a linear combination of $(\mathbf{s}_0, \mathbf{s}_1)$ up to some *noise*, which can be corrected using an error-correcting code (ECC) as we will see below.

We can extend this idea into a fully functional protocol as follows, by making additional use of a linear error correcting code (ECC) against short errors in the euclidean norm.[4]

[4]We can use the decoder of [MP12] as an ECC in our construction.

1. R chooses a random $\mathbf{x} \leftarrow_{\$} \mathbb{Z}_q^k$ such that x_i (i.e., the i-th coordinate of $\mathbf{x}$)[5] corresponds to R's input and compute an LWE sample $\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e}$ for an error vector $\mathbf{e}$.
2. Upon receiving $\mathbf{a}'$ from R, S chooses a *short* matrix $\mathbf{R}$ and computes $\mathbf{s}_0 = \mathbf{a}'\mathbf{R}$ and $\mathbf{s}_1 = \mathbf{a}_i\mathbf{R}$. Moreover, it sends $\mathbf{C} = \mathbf{A}_{-i}\mathbf{R}$, $\mathbf{t}_0 = \mathbf{s}_0 + \hat{\mathbf{z}}_0$ and $\mathbf{t}_1 = \mathbf{s}_1 + \hat{\mathbf{z}}_1$ where $(\hat{\mathbf{z}}_0, \hat{\mathbf{z}}_1)$ is an encoding of S's inputs $(\mathbf{z}_0, \mathbf{z}_1)$.
3. Upon receiving $\mathbf{C}$, R computes
$$\tilde{\mathbf{y}} = \mathbf{x}_{-i}\mathbf{C} - (\mathbf{t}_0 - x_i\mathbf{t}_1) = (\hat{\mathbf{z}}_0 + x_i\hat{\mathbf{z}}_1) + \mathbf{e}'.$$
Then, it decodes $\tilde{\mathbf{y}}$ to obtain $\mathbf{y}$.

Security. Security against a semi-honest sender can be routinely established from the LWE assumption.

To argue that the protocol is secure against a semi-honest receiver, we can show that conditioned on $\mathbf{y} = \mathbf{z}_0 + x_i\mathbf{z}_1$ the values $(\mathbf{z}_0, \mathbf{z}_1)$ are statistically hidden from the view of R. This argument relies on the fact that $\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e}$ is *well-formed* and that $\mathbf{C} = \mathbf{A}_{-i}\mathbf{R}$ is statistically close to uniform even given the residual term $\mathbf{e}\mathbf{R}$ (which can be established via the *Partial Smoothing Lemma* [BD18]).

A closer inspection at the protocol reveals that, in fact, security holds even against an unbounded *malicious* receiver. In order to prove UC-security, we need to construct a simulator which extracts R's input [Can01]. By generating a matrix $\mathbf{A}$ in the CRS along with a lattice-trapdoor[6] $\mathsf{td}_{\mathbf{A}}$ in the sense of [GPV08, MP12], the simulator can extract $\mathbf{x}$ from $\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e}$, provided that $\mathbf{e}$ is *short*.

If extraction fails, meaning that $\mathbf{e}$ is *too large*, we have to consider two different cases. Either i) the vector $\mathbf{a}'$ is *not close* to the row-span of $\mathbf{A}$, or ii) $\mathbf{a}'$ is *close* to the row-span of $\mathbf{A}$. In the first (and easier) case we will be able to rely on the fact that the lattice $\Lambda_q(\mathbf{A}')$ generated by the row-span of $\mathbf{A}' = \begin{pmatrix}\mathbf{A}\\ \mathbf{a}'\end{pmatrix}$ has no short vectors. The *Smoothing Lemma* [MR07] guarantees that if $\mathbf{R}$ is sampled according to a discrete Gaussian distribution (with a properly chosen parameter), then the distribution of the product $\mathbf{A}'\mathbf{R}$ is statistically close to uniform. Thus, in this case, R just obtains random garbage and the simulation succeeds.

We now turn our attention to the most interesting case of our security proof where extraction fails but $\mathbf{a}'$ is close to the row-span of $\mathbf{A}$. First, note that if

[5]Here, the index i can be adversarially chosen by the receiver since the security does not depend on the distribution of i. We remark that i can also be fixed (e.g., $i = 1$). However, we choose to use this notation because we later want to batch several receiver's input into a single message of the same form.

[6]Recall that a lattice trapdoor $\mathsf{td}_{\mathbf{A}}$ (as in [GPV08, MP12]) can be generated along with a matrix $\mathbf{A}$ such that it allows to invert LWE samples. That is, given $\mathbf{x}\mathbf{A} + \mathbf{e}$ for a short vector $\mathbf{e}$, $\mathsf{td}_{\mathbf{A}}$ allows to recover $\mathbf{x}$.

this is the case, then $\mathbf{a}'$ must be of the form $\mathbf{a}' = \alpha(\mathbf{xA} + \mathbf{e}')$ for $\alpha \in \mathbb{Z}_q$ and a short error $\mathbf{e}'$. If the modulus q is polynomial in the security parameter, then the extraction technique of [GPV08, PVW08], which tries to decode every $q-1$ multiples of $\mathbf{a}'$, can be used. However, this would restrict the OLE functionality since the protocol could only be performed for fields $\mathbb{Z}_q$ of polynomial size. Hence, we devise a method to extract $\mathbf{x}$ successfully that works *independently* of the modulus q.

Extraction of the receiver's input. Recall that $\mathtt{td}_{\mathbf{A}}$ is a short square matrix $\mathbf{T} \in \mathbb{Z}_q^{n\times n}$ such that $\mathbf{AT} = 0$ [MP12]. If we multiply $\mathbf{a}'$ by $\mathbf{T}$ we obtain

$$\mathbf{a}'\mathbf{T} = \alpha(\mathbf{xA} + \mathbf{e}')\mathbf{T} = \alpha \cdot \mathbf{e}'\mathbf{T} = \alpha \cdot \mathbf{f} = \mathbf{y}$$

where $\mathbf{f} = \mathbf{e}'\mathbf{T}$ is a short vector. Thus, it suffices to build an algorithm that recovers $\mathbf{f} = (f_1, \ldots, f_n) \in \mathbb{Z}_q^n$ given $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{Z}_q^n$.

From the equation $\mathbf{y} = \alpha\mathbf{f}$, we have that $f_j - (y_j/y_1)f_1 = 0$ for $j = 2, \ldots, n$. Therefore, it is enough to find f_1, since all other coordinates of $\mathbf{f}$ are determined by it.

To find the first coordinate f_1, we rely on the fact that solving the Shortest Vector Problem (SVP) in a two-dimensional lattice can actually be done in polynomial time (and independently of the modulus q) [LP94]. Consider the lattice generated by the two dimensional vector $\mathbf{b}_j = (-y_j/y_1, 1)$. By applying a SVP solver, we are able to find the shortest solution $\mathbf{g}_j = (g_j^{(1)}, g_j^{(2)})$ such that $\mathbf{b}_j \cdot \mathbf{g}_j = 0$. Observe that f_1 must be a multiple of $g_j^{(1)}$ for all $j = 2, \ldots, n$ (otherwise, $\mathbf{g}_j$ would not be the shortest solution of the SVP instance). Hence, f_1 can be computed by taking the least common multiple of $g_1^{(1)}, \ldots, g_n^{(1)}$.

Comparison with previous extraction techniques. All in all, the algorithm constructed above yields if a target vector $\mathbf{a}'$ is close to the row-span of $\mathbf{A}$ or not, given a matrix $\mathbf{A}$ and the corresponding trapdoor $\mathtt{td}_{\mathbf{A}}$. This algorithm greatly improves upon previous extraction techniques such as the one from [PVW08] and we believe that it is of independent interest.

Recall that in [PVW08], the extraction is done by checking if any multiple of $\mathbf{a}'$ is invertible. This immediately imply a polynomial bound on the modulus q, otherwise the simulation would not be efficient. In our case, we avoid this restriction on q and are able to extract the receiver's input *independently* of the modulus q. In the OLE setting, this is of utmost importance since we want to be able to perform OLE over any field $\mathbb{Z}_q$.

The recent work of Quach [Qua20] devised an extraction method for super-polynomial modulus q by using Hash Proof Systems (HPS). However, the use of HPS in the lattice setting comes at the cost of efficiency.[7]

[7]Despite numerous efforts, HPS in the lattice setting fall short in efficiency when comparing to their group-based counterpart.

2.2 Batch OLE.

We define a new OLE functionality that we called *batch OLE.* In this functionality, the receiver *commits* to a batch of inputs $\{x_i\}_{i\in[k']}$. Later, the sender can send its inputs $(\mathbf{z}_0, \mathbf{z}_1)$ together with an index $j \in [k']$ corresponding to which input the receiver is using in this execution of the OLE. The receiver outputs the linear combination $\mathbf{z}_0 + x_j\mathbf{z}_1$.

A slight variant of the OLE protocol described above implements this functionality: In the first round, R chooses $\mathbf{x} \in \mathbb{Z}_q^k$ such that, say, the first k' coordinates correspond to its inputs $(x_1, \ldots, x_{k'})$, and computes $\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e}$. Then, along with its message, the sender S sends a position $j \in [k']$ to indicate which is the receiver's input being used in that execution of the protocol.

This variant improves in communication efficiency since, if R has several inputs, the parties don't need to run the protocol multiple times in parallel. Instead, they can just use the same message $\mathbf{a}'$ for several inputs of the receiver, as long as the LWE assumption holds for dimension $k - k'$.

2.3 Comparison with Previous Works.

The idea of removing a row to a matrix $\mathbf{A}$ to *hide* something and then *recovering* it during *decryption* was already used in a previous work [DGHM18] to construct Hash with Encryption. However, in [DGHM18], the value to hash is chosen selectively by the adversary and thus security follows easily from the Extended LWE assumption [AP12].

Our case presents much more subtle technical challenges since the value $\mathbf{a}'$ sent by the receiver is chosen *after* seeing the matrix $\mathbf{A}$, and thus, it has an *adaptive* flavor.

2.4 Extending to Malicious Adversaries

In the scheme above, it is information-theoretically impossible for a UC-simulator to extract the sender's input. In this section, we show how to modify the scheme to support UC-security against malicious senders.

In a nutshell, the idea to make the protocol secure against corrupted senders is to use a *cut-and-choose*-style approach using a two-round OT protocol, which exists under various assumptions [PVW08, DGH⁺20]. Using the OT, the receiver is able to check if the matrices $\mathbf{C}_j = \mathbf{A}_{-i}\mathbf{R}_j$ sent by the sender are well-formed. More precisely, our protocol works as follows:

1. R computes $\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e}$ as in the previous protocol. Additionally, it runs λ instances of the OT in parallel (playing the role of the receiver), with input bits $(b_1, \ldots, b_\lambda)$, where half of them are equal to 0 and the remaining are equal to 1; and sends the first messages of each instance.

2. For $j \in [\lambda]$, S computes the matrix $\mathbf{C}_j = \mathbf{A}_{-i}\mathbf{R}_j$ for a short matrix $\mathbf{R}_j$. It chooses two random vectors $(\mathbf{u}_{0,j}, \mathbf{u}_{1,j})$ and inputs two messages $(M_{0,j}, M_{1,j})$ in the OT, where $M_{0,j} = \mathbf{R}_j$ and $M_{1,j} = (\mathbf{t}_{0,j}, \mathbf{t}_{1,j}, \mathbf{u}_{0,j} +$

$\mathbf{z}_0, \mathbf{u}_{1,j} + \mathbf{z}_1)$ where $\mathbf{t}_{0,j} = \mathbf{a}'\mathbf{R}_j + \tilde{\mathbf{e}}_j + \hat{\mathbf{u}}_{0,j}$ and $\mathbf{t}_{1,j} = \mathbf{a}_i\mathbf{R}_j + \hat{\mathbf{u}}_{1,j}$, where $\hat{\mathbf{u}}_{0,j}$ and $\hat{\mathbf{u}}_{1,j}$ are the encodings of $\mathbf{u}_{0,j}$ and $\mathbf{u}_{0,j}$, respectively.

3. When $b_j = 0$, R can check that the matrix $\mathbf{C}_j$ is indeed well-formed. When $b_j = 1$, R can use $M_{1,j}$ to compute the *same* linear combination $\mathbf{w} = \mathbf{z}_0 + x_i\mathbf{z}_1$ for every position j (it aborts if there is at least one different from the others).

Security against an unbouded receiver in the OT-hybrid model essentially follows the same reasoning as in the previous protocol.

We now argue how we can build the simulator Sim against a corrupted sender. By simulating the OT functionality, the simulator Sim can extract the sender's input using a single position j for which S inputs the *right* messages $M_{0,j}$ and $M_{1,j}$. This event always happens, except with negligible probability. Moreover, if R accepts the messages $M_{1,j}$, when $b_j = 1$, then this means that S input the same pair $(\mathbf{z}_0, \mathbf{z}_1)$ in these positions given that the LWE assumption holds.

The main drawback of this approach is that we loose reusability.[8] It is trivial to see that this extraction strategy fails if S knows which are the bits that R sends to the OT functionality. Moreover, after several executions of the protocol, S is able to correctly guess the values of these bits. In fact, it is known that reusability is impossible in the OT-hybrid model [CDI+19].

Instantiating the OT functionality. When we instantiate our protocol with the OT schemes from [PVW08, Qua20] we obtain several nice properties for the OLE scheme, namely: i) The protocol is still two-round; ii) the protocol preserves statistical security against a corrupted receiver (since the OT of [PVW08, Qua20] is also statistically secure against a corrupted receiver); and iii) the OLE scheme is secure based solely on the LWE assumption.

3 Preliminaries

Throughout this work, λ denotes the security parameter and PPT stands for "probabilistic polynomial-time".

Let $\mathbf{A} \in \mathbb{Z}_q^{k\times n}$ and $\mathbf{x} \in \mathbb{Z}_q^n$. We denote by $\mathbf{A}_{-i}$ the matrix $\mathbf{A}$ with the i-th row removed. Similarly, $\mathbf{x}_{-i}$ denotes the vector $\mathbf{x}$ with the i-th coordinate removed. Moreover, $\|\mathbf{x}\|$ denotes the usual ℓ_2 norm of a vector $\mathbf{x}$. For a vector $\mathbf{b} \in \{0,1\}^k$, we denote its weight, that is the number of non-null coordinates, by $wt(\mathbf{b})$.

If S is a (finite) set, we denote by $x \leftarrow_\$ S$ an element $x \in S$ sampled according to a uniform distribution. Moreover, we denote by $U(S)$ the uniform distribution over S. If D is a distribution over S, $x \leftarrow_\$ D$ denotes an element $x \in S$ sampled according to D. If $\mathcal{A}$ is an algorithm, $y \leftarrow \mathcal{A}(x)$ denotes the output y after running $\mathcal{A}$ on input x.

[8]Remark that reusability is trivially achieved when the sender is semi-honest.

A negligible function $\mathsf{negl}(n)$ in n is a function that vanishes faster than the inverse of any polynomial in n.

Given two distributions D_1 and D_2, we say that they are ε-statistically indistinguishable, denoted by $D_1 \approx_\varepsilon D_2$, if the statistical distance is at most ε.

Error-Correcting Codes. We define Error-Correcting Codes (ECC). An ECC over $\mathbb{Z}_q$ is composed by the following algorithms $\mathsf{ECC}_{\ell,k,\delta} = (\mathsf{Encode}, \mathsf{Decode})$ such that: i) $\mathbf{c} \leftarrow \mathsf{Encode}(\mathbf{m})$ takes as input a message $\mathbf{m} \in \mathbb{Z}_q^\ell$ and outputs a codeword $\mathbf{c} \in \mathbb{Z}_q^k$; ii) $\mathbf{m} \leftarrow \mathsf{Decode}(\tilde{\mathbf{c}})$ takes as input corrupted codeword $\tilde{\mathbf{c}} \in \mathbb{Z}_q^k$ and outputs a message $\mathbf{m} \in \mathbb{Z}_q^\ell$ if $\|\tilde{\mathbf{c}} - \mathbf{c}\| \leq \delta$ where $\mathbf{c} \leftarrow \mathsf{Encode}(\mathbf{m})$. In this case, we say that ECC corrects up to δ errors. We say that ECC is linear if any linear combination of codewords of ECC is also a codeword of ECC.

An example of such code is the primitive lattice of [MP12] which allows for efficient decoding and fulfills all the properties that we need.

3.1 Universal Composability

UC-framework [Can01] allows to prove security of protocols even under arbitrary composition with other protocols. Let $\mathcal{F}$ be a functionality, π a protocol that implements $\mathcal{F}$ and $\mathcal{Z}$ be a environment, an entity that oversees the execution of the protocol in both the real and the ideal worlds. Let $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}$ be a random variable that represents the output of $\mathcal{Z}$ after the execution of $\mathcal{F}$ with adversary Sim. Similarly, let $\mathsf{REAL}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$ be a random variable that represents the output of $\mathcal{Z}$ after the execution of π with adversary $\mathcal{A}$ and with access to the functionality $\mathcal{G}$.

A protocol π *UC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model* if for every PPT adversary $\mathcal{A}$ there is a PPT simulator Sim such that for all PPT environments $\mathcal{E}$, the distributions $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}$ and $\mathsf{REAL}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable.

In this work, we only consider *static* adversaries. That is, parties involved in the protocol are corrupted at the beginning of the execution.

We now present the ideal functionalities that we will use in this work.

CRS functionality. This functionality generates a crs and distributes it between all the parties involved in the protocol. Here, we present the ideal functionality as in [PVW08].

$\mathcal{G}_{\mathsf{CRS}}$ functionality

Parameters: An algorithm D.

- Upon receiving $(\mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j)$ from P_i, $\mathcal{G}_{\mathsf{CRS}}$ runs $\mathsf{crs} \leftarrow \mathsf{D}(1^\kappa)$ and returns $(\mathsf{sid}, \mathsf{crs})$ to P_i.
- Upon receiving $(\mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j)$ from P_j, $\mathcal{G}_{\mathsf{CRS}}$ returns $(\mathsf{sid}, \mathsf{crs})$ to P_j.

OT functionality. Oblivious Transfer (OT) can be seen as a particular case of OLE. We show the ideal OT functionality below.

$\mathcal{F}_{\mathsf{OT}}$ functionality

Parameters: $\mathsf{sid} \in \mathbb{N}$ known to both parties.

- Upon receiving $(\mathsf{sid}, (M_0, M_1))$ from S, $\mathcal{F}_{\mathsf{OT}}$ stores (M_0, M_1) and ignores future messages from S with the same sid;
- Upon receiving $(\mathsf{sid}, b \in \{0,1\})$ from R, $\mathcal{F}_{\mathsf{OT}}$ checks if it has recorded $(\mathsf{sid}, (M_0, M_1))$. If so, it returns (sid, M_b) to R and $(\mathsf{sid}, \mathsf{receipt})$ to S, and halts. Else, it sends nothing, but continues running.

OLE functionality. We now present the OLE functionality. This functionality involves two parties: the sender S and the receiver R.

$\mathcal{F}_{\mathsf{OLE}}$ functionality

Parameters: $\mathsf{sid}, q, k \in \mathbb{N}$ and a finite field $\mathbb{F}$ known to both parties.

- Upon receiving $\left(\mathsf{sid}, (\mathbf{a}, \mathbf{b}) \in \mathbb{F}^k \times \mathbb{F}^k\right)$ from S, $\mathcal{F}_{\mathsf{OLE}}$ stores $(\mathbf{a}, \mathbf{b})$ and ignores future messages from S with the same sid;
- Upon receiving $(\mathsf{sid}, x \in \mathbb{F})$ from R, $\mathcal{F}_{\mathsf{OLE}}$ checks if it has recorded $(\mathsf{sid}, (\mathbf{a}, \mathbf{b}))$. If so, it returns $(\mathsf{sid}, \mathbf{z} = \mathbf{a} + x\mathbf{b})$ to R and $(\mathsf{sid}, \mathsf{receipt})$ to S, and halts. Else, it sends nothing but continues running.

Batch OLE functionality. Here we define a batch version of the functionality defined above. In this functionality, the receiver inputs several OLE inputs at the same time. The sender can then input an affine function together with an index corresponding to which input the receiver should receive the linear combination.

$\mathcal{F}_{\mathsf{bOLE}}$ functionality

Parameters: $\mathsf{sid}, q, k, k' \in \mathbb{N}$ and a finite field $\mathbb{F}$ known to both parties.

- Upon receiving $\left(\mathsf{sid}, \{(\mathbf{a}_i, \mathbf{b}_i)\}_{i \in [k']} \in \mathbb{F}^k \times \mathbb{F}^k\right)$ from S, $\mathcal{F}_{\mathsf{bOLE}}$ stores $\{(\mathbf{a}_i, \mathbf{b}_i)\}_{i \in [k']}$ and ignores future messages from S with the same sid;
- Upon receiving $(\mathsf{sid}, \{x_i\}_{i \in [k']})$ from R, where $x_i \in \mathbb{F}$, $\mathcal{F}_{\mathsf{bOLE}}$ checks if it has recorded $\left(\mathsf{sid}, \{(\mathbf{a}_i, \mathbf{b}_i)\}_{i \in [k']}\right)$. If so, it returns $\left(\mathsf{sid}, \{\mathbf{z}_i = \mathbf{a}_i + x_i\mathbf{b}_i\}_{i \in [k']}\right)$ to R and $(\mathsf{sid}, \mathsf{receipt})$ to S, and halts. Else, it sends nothing but continues running.

3.2 Lattices and Hardness Assumptions

Notation. Let $\mathbf{B} \in \mathbb{R}^{k \times n}$ be a matrix. We denote the lattice generated by $\mathbf{B}$ by $\Lambda = \Lambda(\mathbf{B}) = \{\mathbf{x}\mathbf{B} : \mathbf{x} \in \mathbb{Z}^k\}$.[9] The dual lattice Λ^* of a lattice Λ is defined by $\Lambda^* = \{\mathbf{x} \in \mathbb{R}^n : \forall y \in \Lambda, \mathbf{x} \cdot \mathbf{y} \in \mathbb{Z}\}$. It holds that $(\Lambda^*)^* = \Lambda$.

We denote by $\gamma\mathcal{B}$ the ball of radius γ centered on zero. That is

$$\gamma\mathcal{B} = \{\mathbf{x} \in \mathbb{Z}^n : \|\mathbf{x}\| \leq \gamma\}.$$

A lattice Λ is said to be q-ary if $(q\mathbb{Z})^n \subseteq \Lambda \subseteq \mathbb{Z}^n$. For every q-ary lattice Λ, there is a matrix $\mathbf{A} \in \mathbb{Z}_q^{k \times n}$ such that

$$\Lambda = \Lambda_q(\mathbf{A}) = \{y \in \mathbb{Z}^n : \exists \mathbf{x} \in \mathbb{Z}_q^k, \mathbf{y} = \mathbf{x}\mathbf{A} \mod q\}.$$

The orthogonal lattice $\Lambda_q^\perp$ is defined by $\{\mathbf{y} \in \mathbb{Z}_q^n : \mathbf{A}\mathbf{y}^T = 0 \mod q\}$. It holds that $\frac{1}{q}\Lambda_q^\perp = \Lambda_q^*$

Let $\rho_s(\mathbf{x})$ be probability distribution of the Gaussian distribution over $\mathbb{R}^n$ with parameter s and centered in 0. We define the discrete Gaussian distribution $D_{S,s}$ over S and with parameter s by the probability distribution $\rho_s(\mathbf{x})/\rho(S)$ for all $\mathbf{x} \in S$ (where $\rho_s(S) = \sum_{\mathbf{x} \in S} \rho_s(\mathbf{x})$).

For $\varepsilon > 0$, the smoothing parameter $\eta_\varepsilon(\Lambda)$ of a lattice Λ is the least real $\sigma > 0$ such that $\rho_{1/\sigma}(\Lambda^* \setminus \{0\}) \leq \varepsilon$ [MR07].

Useful Lemmata. The following lemmas are well-known results on discrete Gaussians over lattices.

Lemma 1 ([Ban93]). *Let $\sigma > 0$ and $\mathbf{x} \leftarrow_\$ D_{\mathbb{Z}^n,\sigma}$. Then we have that*

$$\Pr\left[\|\mathbf{x}\| \geq \sigma\sqrt{n}\right] \leq \mathsf{negl}(n)\,.$$

The next lemma is a consequence of the smoothing lemma [MR07] and it tells us that $\mathbf{A}\mathbf{e}^T$ is uniform, when $\mathbf{e}$ is sampled from a discrete Gaussian with a proper choice of parameters.

Lemma 2 ([GPV08]). *Let $q \in \mathbb{N}$ and $\mathbf{A} \in \mathbb{Z}_q^{k \times n}$ be a matrix whose columns generate $\mathbb{Z}_q^k$. Moreover, let $\varepsilon \in (0, 1/2)$ and $\sigma \geq \eta_\varepsilon(\Lambda_q^\perp(\mathbf{A}))$. Then, for $\mathbf{e} \leftarrow_\$ D_{\mathbb{Z}^m,\sigma}$,*

$$\mathbf{A}\mathbf{e}^T \mod q \approx_{2\varepsilon} \mathbf{u}^T \mod q$$

where $\mathbf{u} \leftarrow_\$ \mathbb{Z}_q^k$.

The partial smoothing lemma tells us that the famous smoothing lemma [MR07] still holds even given a small leak.

Lemma 3 (Partial Smoothing [BD18]). *Let $q \in \mathbb{N}$, $\gamma > 0$ be a real number, $\mathbf{A} \in \mathbb{Z}_q^{k \times n}$ and $\sigma, \varepsilon > 0$ be such that $\rho_{q/\sigma}(\Lambda_q(\mathbf{A}) \setminus \gamma\mathcal{B}) \leq \varepsilon$. Moreover, let $\mathbf{D} \in \mathbb{Z}_q^{m \times k}$ be a full-rank matrix with $\Lambda_q^\perp(\mathbf{D}) = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{x} \cdot \mathbf{y} = 0, \forall \mathbf{y} \in \Lambda_q(\mathbf{A}) \cap \gamma\mathcal{B}\}$. Then we have that*

$$\mathbf{A}\mathbf{x}^T \mod q \approx_\varepsilon \mathbf{A}(\mathbf{x} + \mathbf{u})^T \mod q$$

where $\mathbf{x} \leftarrow_\$ D_{\mathbb{Z}^n,\sigma}$ and $\mathbf{u} \leftarrow_\$ \Lambda_q^\perp(\mathbf{D}) \mod q$.

[9] The matrix $\mathbf{B}$ is called a basis of $\Lambda(\mathbf{B})$.

LWE Assumption. The Learning with Errors assumption was first presented in [Reg05]. The assumption roughly states that it should be hard to solve a set linear equations by just adding a little noise to it.

Definition 1 (Learning with Errors)**.** *Let $q, k \in \mathbb{N}$ where $k \in \mathsf{poly}(\lambda)$, $\mathbf{A} \in \mathbb{Z}_q^{k\times n}$ and $\beta \in \mathbb{R}$. For any $n = \mathsf{poly}(k \log q)$, the $\mathsf{LWE}_{k,\beta,q}$ assumption holds if for every PPT algorithm $\mathcal{A}$ we have*

$$|\Pr[1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{sA} + \mathbf{e})] - \Pr[1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{y})]| \leq \mathsf{negl}(\lambda)$$

for $\mathbf{s} \leftarrow_{\$} \{0,1\}^k$, $\mathbf{e} \leftarrow_{\$} D_{\mathbb{Z}^n,\beta}$ and $\mathbf{y} \leftarrow_{\$} \{0,1\}^n$.

Regev proved in [Reg05] that there is a (quantum) worst-case to average-case reduction from some problems on lattices which are believed to be hard even in the presence of a quantum computer.

Trapdoors for Lattices. Recent works [GPV08, MP12] have presented trapdoors functions based on the hardness of LWE.

Lemma 4 ([GPV08, MP12])**.** *Let $\tau(k) \in \omega\left(\sqrt{\log k}\right)$ be a function. There is a pair of algorithms $(\mathsf{TdGen}, \mathsf{Invert})$ such that if $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TdGen}(q, k)$ then:*

- *$\mathbf{A} \in \mathbb{Z}_q^{k\times n}$ where $n \in \mathcal{O}(k \log q)$ is a matrix whose distribution is 2^{-k} close to the uniform distribution over $\mathbb{Z}_q^{k\times n}$.*
- *For any $\mathbf{s} \in Z_q^k$ and $\mathbf{e} \in \mathbb{Z}_q^n$ such that $\|\mathbf{e}\| < q/(\sqrt{n}\tau(k))$, we have that*

$$\mathbf{s} \leftarrow \mathsf{Invert}(\mathsf{td}, \mathbf{sA} + \mathbf{e}).$$

In the lemma above, td corresponds to a *short* matrix $\mathbf{T} \in \mathbb{Z}_q^{n\times n}$ (that is, $\max_{i\in[n]}\{\mathbf{t}^{(i)}\} < B$, where $\mathbf{t}^{(i)}$ is the i-th column of $\mathbf{T}$, for some $B \in \mathbb{N}$) such that $\mathbf{AT} = 0$ and $\mathbf{T}^{-1}$ is easily computed. To invert a sample of the form $\mathbf{y} = \mathbf{sA} + \mathbf{e}$, we simply compute $\mathbf{yT} = \mathbf{sAT} + \mathbf{eT} = \mathbf{eT}$. The error vector $\mathbf{e}$ can be easily recovered by multiplying by $\mathbf{T}^{-1}$.

Observe that, if $(\mathbf{A}, \mathsf{td}_{\mathbf{A}}) \leftarrow \mathsf{TdGen}(1^\lambda, n, k, q)$, then $\Lambda(\mathbf{A})$ has no *short* vectors. That is, for all $\mathbf{y} \in \Lambda(\mathbf{A})$, then $\|\mathbf{y}\| > B = q/(\sqrt{n}\tau(k))$ [MP12]. If this does not happen, then the algorithm Invert would not output the right $\mathbf{s}$ for a non-negligible number of cases.

4 Finding Short Vectors in a Lattice with a Trapdoor

In this section, we build an algorithm that, given a matrix $\mathbf{A}$ together with the corresponding trapdoor $\mathsf{td}_{\mathbf{A}}$ (in the sense of Lemma 4), we can decide if a vector $\mathbf{a}$ is close to the row-span of $\mathbf{A}$ and even find the corresponding linear combination.

To build our algorithm, we need an additional tool. The following lemma states that, for two-dimensional lattices, we can efficiently find the shortest vector of the lattice.

Lemma 5 ([LP94])**.** *There exists an algorithm* SolveSVP *that takes as input two-dimensional lattice generated by* $\mathbf{a} \in \mathbb{Z}_q^2$ *and outputs the shortest vector* $\mathbf{e} \in \mathbb{Z}^2$ *such that* $\mathbf{a}\mathbf{e}^T = 0$*. Such an algorithm runs it time* $\mathcal{O}(\log q)$*.*

The following algorithm allows us to solve an equation of the form $r \cdot \mathbf{e} = \mathbf{y}$ for a short vector $\mathbf{e}$. This algorithm will be instrumental to the main result of this section.

Lemma 6. *Let* $q \in \mathbb{N}$ *be a prime number. There exists an algorithm* RecoverError *that takes as input a vector* $\mathbf{y} \in \mathbb{Z}_q^n$ *and a bound* B*. The vector* $\mathbf{y}$ *is of the form* $\mathbf{y} = r\mathbf{e}$ *for some* $r \in \mathbb{Z}_q$ *and* $\mathbf{e} \in \mathbb{Z}_q^n$ *with* $\|\mathbf{e}\| \leq B$*. It outputs* $(r, \mathbf{e})$*. The algorithm runs in polynomial time in* n*.*

Proof. To prove the lemma above, we first present how the algorithm works.

Construction 1. *Let* $q \in \mathbb{N}$ *be a prime number and* $n = \mathsf{poly}(\lambda)$*. For the sake of simplicity, we assume that no coordinate of* $\mathbf{y}$ *is equal to zero.*

$\mathsf{RecoverError}_{q,n}(\mathbf{y}, B)$**:**

- *Parse* $\mathbf{y} \in \mathbb{Z}_q^n$ *as* $(y_1, \dots, y_n)$ *and* $B > 0$*. If* $\|\mathbf{y}\| \leq B$ *output* $\mathbf{y}$*.*
- *For all* $i = 2, \dots, n$*, compute* $v_i = y_i/y_1$*. Consider the lattice* $\Lambda_q(\mathbf{a}_i)$ *generated by the vector* $\mathbf{a}_i = (-v_i, 1)$*. Apply* $\mathsf{SolveSVP}(\mathbf{a}_i)$ *to obtain* $\mathbf{e}_i \in \mathbb{Z}_q^2$ *such that* $\mathbf{e}_i = (e_i^{(1)}, e_i^{(2)}) \in \Lambda_q^\perp(\mathbf{a}_i)$*.*
- *Compute* $e_1 = \mathsf{lcm}(e_1^{(1)}, \dots, e_n^{(1)})$*, where* lcm *is the least common multiple function.*
- *For all* $i = 2, \dots, n$*, compute* $e_i = v_i e_1$*.*
- *If* $\|\mathbf{e}\| < B$*, output* $(r, \mathbf{e})$ *where* $\mathbf{e} = (e_1, \dots, e_n)$ *and* $r = y_1/e_1$*. Else, output* $\perp$*.*

We now proceed to analyze the algorithm above. Again, we assume that all coordinates of $\mathbf{y}$ are non-zero. If this is not the case, then we set $e_i = 0$ for all i where $y_i = 0$ and apply the algorithm to the remaining coordinates. Assume that $\mathbf{y} = r \cdot \mathbf{e}$, that is, $(y_1, \dots, y_n) = (re_1, \dots, re_n)$. If we divide all coordinates $i \geq 2$ by the first coordinate re_i, we obtain the following equations

$$\begin{gathered} e_2 - (y_2/y_1)e_1 = 0 \\ \vdots \\ e_n - (y_n/y_1)e_1 = 0. \end{gathered} \tag{1}$$

Set $v_i = y_i/y_1$ and consider the vector $\mathbf{a}_i = (-v_i, 1)$. After applying SolveSVP to $\mathbf{a}_i$ we obtain the shortest vector $\mathbf{e}'_i = (e_i^{(1)}, e_i^{(2)})$ such that $\mathbf{a}_i\mathbf{e}'_i = 0$. Note that, since $\mathbf{a}_i$ has dimension 2, then SolveSVP runs in time $\mathcal{O}(\log q)$ by Lemma 5.

We claim that $e_1 = t_i e_i^{(1)}$ for all i and $t_i \in \mathbb{Z}$, that is, e_1 is a multiple of $e_i^{(1)}$. To see this note that $\mathbf{e}'_i$ is sampled from $\Lambda_q^{\perp}(\mathbf{a}_i)$. Thus

$$\begin{aligned}
&e_i - v_i e_1 = e_i^{(2)} - v_i e_i^{(1)} = 0\\
&e_1/e_i^{(1)} = e_i/e_i^{(2)}\\
&e_1 = (e_i/e_i^{(2)})e_i^{(1)}.
\end{aligned}$$

Given that $e_1, e_i, e_i^{(1)}, e_i^{(2)} \in \mathbb{Z}$ and that $e_i^{(1)}$ and $e_i^{(2)}$ are coprime (otherwise, it would not be the shortest solution) then we must have $(e_i^{(2)}/e_i) \in \mathbb{Z}$.

Therefore, taking the least common multiple of $e_i^{(1)}$'s yields the shortest possible value e_1 that fulfills all equations.

By the system of equations 1, it is enough to find e_1 to recover the whole vector $\mathbf{e}$ since all other coordinates are determined by e_1. $\square$

We now present the main result of this section. The lemma states that we can decide if a given vector $\mathbf{a}$ is close to the row-span of $\mathbf{A}$, if $\mathbf{A}$ is generated together with a trapdoor.

Lemma 7. *Let $q \in \mathbb{N}$ be a prime number and* TdGen *be the algorithm from 4. Let $(\mathbf{A}, \mathsf{td}_{\mathbf{A}}) \leftarrow \mathsf{TdGen}(q, k)$. There exists an algorithm* InvertCloseVector *that takes as input a trapdoor $\mathsf{td}_{\mathbf{A}}$, a vector $\mathbf{a} \in \mathbb{Z}_q^n$ and a bound $B > 0$. If there are $\mathbf{x} \in \mathbb{Z}_q^k$ and $r \in \mathbb{Z}_q$ such that $\mathbf{xA} + r\mathbf{a} = \mathbf{e}$ for some $\mathbf{e} \in \mathbb{Z}_q^n$ such that $\|\mathbf{e}\| < B$, the algorithm outputs $(\mathbf{x}, r, \mathbf{e})$. Else, it outputs $\perp$.*

Proof. We present the construction of the new algorithm InvertCloseVector

Construction 2. *Let $(\mathbf{A}, \mathsf{td}_{\mathbf{A}}) \leftarrow$* TdGen *and let* RecoverError *be the algorithm from Lemma 6.*

$\mathsf{InvertCloseVector}(\mathsf{td}_{\mathbf{A}}, \mathbf{a}, B)$:

- *Parse $\mathsf{td}_{\mathbf{A}} = \mathbf{T} \in \mathbb{Z}_q^{n\times n}$, $\mathbf{a} \in \mathbb{Z}_q^n$ and $B > 0$.*
- *Compute $\mathbf{z} = \mathbf{aT}$.*
- *Apply $(r, \mathbf{e}') \leftarrow \mathsf{RecoverError}_{q,n}(\mathbf{z}, B)$. Compute $\mathbf{e} = \mathbf{e}'\mathbf{T}^{-1}$*
- *Check if $\|\mathbf{e}\| < B$ and recover $\mathbf{x}'$ such that $\mathbf{x}'\mathbf{A} + r \cdot \mathbf{e}$. Set $\mathbf{x} = r^{-1}\mathbf{x}$*
- *If $\|\mathbf{e}\| > B$ output $\perp$. Else, output $(\mathbf{x}, r, \mathbf{e})$.*

We have two cases to consider: Either the vector $\mathbf{a}$ is close to the row-span of $\mathbf{A}$ or it is not.

We begin by analyzing the first case. If $\mathbf{a}$ is close to the row-span of $\mathbf{A}$, then there exists $\mathbf{x}$ and α such that $\mathbf{xA} + \alpha\mathbf{a} = \mathbf{e}$ for some $\mathbf{e} \in \mathbb{Z}_q^n$ such that $\|\mathbf{e}\| < B$. This means that $\mathbf{a}$ is a multiple of a point close to the row-span of $\mathbf{A}$, i.e., $\mathbf{a} = \mathbf{x}'\mathbf{A} + r \cdot \mathbf{e}$ for some $\mathbf{x}'$ and r.

Parsing $\mathsf{td}_{\mathbf{A}}$ as $\mathbf{T} \in \mathbb{Z}_q^{n\times n}$ and multiplying by $\mathbf{a}$, we obtain

$$\mathbf{aT} = \mathbf{x}'\mathbf{AT} + r\mathbf{eT} = r.\mathbf{e}'$$

where $\mathbf{e}' = \mathbf{eT}$ such that $\|\mathbf{e}'\| < B'$ for some $B' > 0$ and where the last equality holds because $\mathbf{AT} = 0$.

Now, by Lemma 6, we recover $(r, \mathbf{e}')$ after running $\mathsf{RecoverError}$. From this, we can recover $\mathbf{e} = \mathbf{e}'\mathbf{T}^{-1}$ and $\mathbf{x} = r^{-1}\mathbf{x}'$, where $\mathbf{x}'\mathbf{A} = \mathbf{a} - r\mathbf{e}$. $\square$

5 Oblivious Linear Evaluation Secure Against a Corrupted Receiver

In this section, we present a semi-honest protocol for OLE based on the hardness of the LWE assumption. The protocol implements functionality $\mathcal{F}_{\mathsf{OLE}}$ defined in Section 3.

5.1 Protocol

We begin by presenting the protocol.

Construction 3. *The protocol is composed by the algorithms* $(\mathsf{GenCRS}, \mathsf{R}_1, \mathsf{S}, \mathsf{R}_2)$. *Let* $k, n, \ell, \ell', q \in \mathbb{Z}$ *such that* q *is a prime and* $n = \mathsf{poly}(k \log q)$*, and let* $\beta, \delta, \xi \in \mathbb{R}$ *such that* $\frac{q}{\sqrt{n}\tau(k)} > \beta$ *(where* $\tau(k) = \omega(\sqrt{\log k})$ *as in Lemma 4),* $\delta > \beta > 1$ *and* $\beta > q/\delta$*. We present the protocol in full detail.*

$\mathsf{GenCRS}(1^\lambda)$**:**

- *Sample* $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{k\times n}$.
- *Choose a linear ECC* $\mathsf{ECC}_{\ell',\ell,\xi} = (\mathsf{ECC.Encode}, \mathsf{ECC.Decode})$ *over* $\mathbb{Z}_q$.
- *Output* $\mathsf{crs} = (\mathbf{A}, \mathsf{ECC}_{\ell',\ell,\xi})$.

$\mathsf{R}_1\,(\mathsf{crs}, x \in \mathbb{Z}_q)$**:**

- *Parse* crs *as* $(\mathbf{A}, \mathsf{ECC}_{\ell',\ell,\xi})$.
- *Sample* $\mathbf{x} = (x_1, \dots, x_k) \leftarrow_\$ \mathbb{Z}_q^k$ *such that* $x_i = x$ *and a small error vector* $\mathbf{e} \leftarrow_\$ D_{\mathbb{Z}^n,\beta}$*, for a uniformly chosen index* $i \leftarrow_\$ [k]$.
- *Compute* $\mathbf{a}' = \mathbf{xA} + \mathbf{e}$.
- *Output* $\mathsf{ole}_1 = (\mathbf{a}', i)$ *and* $\mathsf{st} = (\mathbf{x}, i)$.

$\mathsf{S}\left(\mathsf{crs}, (\mathbf{z}_0, \mathbf{z}_1) \in (\mathbb{Z}_q^{\ell'})^2, m_{\mathsf{R}}\right)$**:**

- *Parse* crs *as* $(\mathbf{A}, \mathsf{ECC}_{\ell',\ell,\xi})$ *and* ole_1 *as* $(\mathbf{a}', i)$.
- *Sample* $\mathbf{R} \in \mathbb{Z}_q^{n\times\ell}$ *where each column* $\mathbf{r}^{(j)} \leftarrow_{\$} D_{\mathbb{Z}^n,\delta}$ *for* $j \in [\ell]$.
- *Compute* $\mathbf{s}_0, \mathbf{s}_1 \in \mathbb{Z}_q^{\ell}$ *such that* $\mathbf{s}_0 = \mathbf{a}'\mathbf{R}$, $\mathbf{s}_1 = -\mathbf{a}_i\mathbf{R}$, *and* $\mathbf{C} = \mathbf{A}_{-i}\mathbf{R} \in \mathbb{Z}_q^{(k-1)\times\ell}$.
- *Compute* $\mathbf{t}_0 = \mathbf{s}_0 + \mathsf{ECC.Encode}(\mathbf{z}_0)$ *and* $\mathbf{t}_1 = \mathbf{s}_1 + \mathsf{ECC.Encode}(\mathbf{z}_1)$.
- *Output* $\mathsf{ole}_2 = (\mathbf{C}, \mathbf{t}_0, \mathbf{t}_1)$.

$\mathsf{R}_2(\mathsf{crs}, \mathsf{st}, m_{\mathsf{S}})$**:**

- *Parse* crs *as* $(\mathbf{A}, \mathsf{ECC}_{\ell',\ell,\xi})$, ole_2 *as* $(\mathbf{C}, \mathbf{t}_0, \mathbf{t}_1)$ *and* st *as* $(\mathbf{x} \in \mathbb{Z}_q^k, i)$.
- *Compute* $\tilde{\mathbf{y}} = \mathbf{x}_{-i}\mathbf{C} - (\mathbf{t}_0 + x_i\mathbf{t}_1)$ *and then run* $\mathbf{y} \leftarrow \mathsf{ECC.Decode}(\tilde{\mathbf{y}})$. *If* $\mathbf{y} = \perp$, *abort the protocol.*
- *Output* $\mathbf{y} \in \mathbb{Z}_q^{\ell'}$.

5.2 Analysis of the Protocol

Theorem 1 (Correctness)**.** *Let* $\mathsf{ECC}_{\ell',\ell,\xi}$ *be a linear ECC where* $\xi \geq 2\sqrt{\ell}\beta\delta n$. *Then the protocol presented in Construction 3 is correct.*

Proof. To prove correctness, we have to prove that R_2 outputs $\mathbf{z}_0 + x_i\mathbf{z}_1$, where $(\mathbf{z}_0, \mathbf{z}_1)$ is the input of S.

First, note that

$$\begin{aligned}
\mathbf{x}_{-i}\mathbf{C} &= \mathbf{x}_{-i}\mathbf{A}_{-i}\mathbf{R} \\
&= (\mathbf{x}\mathbf{A} - x_i\mathbf{a}_i)\mathbf{R} \\
&= (\mathbf{x}\mathbf{A} + \mathbf{e})\mathbf{R} - (x_i\mathbf{a}_i)\mathbf{R} - (\mathbf{e} + \tilde{\mathbf{e}})\mathbf{R} \\
&= \mathbf{s}_0 + x_i\mathbf{s}_1 + \mathbf{e}'
\end{aligned}$$

where $\mathbf{e}' = -(\mathbf{e} + \tilde{\mathbf{e}})\mathbf{R}$ is a small error vector. The receiver computes

$$\begin{aligned}
\tilde{\mathbf{y}} &= \mathbf{x}_{-i}\mathbf{C} - (\mathbf{t}_0 + x_i\mathbf{t}_1) \\
&= \mathsf{ECC.Encode}(\mathbf{z}_0 + x_i\mathbf{z}_1) + \mathbf{e}'
\end{aligned}$$

where the second equality holds because ECC is linear.

Finally, by Lemma 1, we have that $\|\mathbf{e}\|, \|\tilde{\mathbf{e}}\| \leq \beta\sqrt{n}$. Moreover, if $\mathbf{r}^{(i)}$ is a column of $\mathbf{R}$, then $\left\|\mathbf{r}^{(i)}\right\| \leq \delta\sqrt{n}$. Therefore, each coordinate of $\mathbf{e}'$ has norm at most $(\|\mathbf{e}\| + \|\tilde{\mathbf{e}}\|) \cdot \left\|\mathbf{r}^{(i)}\right\| \leq \beta\delta n$. We conclude that $\|\mathbf{e}'\| \leq 2\sqrt{\ell}\beta\delta n$. Since ECC corrects up to $\xi \geq 2\sqrt{\ell}\beta\delta n$ errors, the output of $\mathsf{ECC.Decode}(\tilde{\mathbf{y}})$ is exactly $\mathbf{z}_0 + x_i\mathbf{z}_1$. ☐

Theorem 2 (Security)**.** *Assume that the* $\mathsf{LWE}_{k,\beta,q}$ *assumption holds,* $\frac{q}{\sqrt{n}\tau(k)} > \beta$ *(where* $\tau(k) = \omega(\sqrt{\log k})$ *as in Lemma 4),* $\delta > \beta > 1$ *and* $\beta > q/\delta$*. The protocol presented in Construction 3 securely realizes the functionality* $\mathcal{F}_{\mathsf{OLE}}$ *in the* $\mathcal{G}_{\mathsf{CRS}}$*-hybrid model against:*

- *a semi-honest sender given that the* $\mathsf{LWE}_{k,\beta,q}$ *assumption holds;*
- *a malicious receiver where security holds statistically.*

Proof. We begin by proving security against a computationally unbounded corrupted receiver.

Simulator for corrupted receiver: We describe the simulator Sim. Let $(\mathsf{TdGen}, \mathsf{Invert})$ be the algorithms described in Lemma 4 and $\mathsf{InvertCloseVector}$ be the algorithm of Lemma 7.

- **CRS generation:** Sim generates $(\mathbf{A}, \mathsf{td}_{\mathbf{A}}) \leftarrow \mathsf{TdGen}(1^\lambda, q, k, n)$. It chooses an ECC $\mathsf{ECC}_{\ell',\ell,\xi}$. It publishes $\mathsf{crs} = (\mathbf{A}, \mathsf{ECC})$ and keeps $\mathsf{td}_{\mathbf{A}}$ to itself.
- Upon receiving a message $\mathbf{a}'$ from R, Sim runs $(\tilde{\mathbf{x}}, \alpha, \mathbf{e}) \leftarrow \mathsf{InvertCloseVector}(\mathsf{td}_{\mathbf{A}}, \mathbf{a}', B)$ where $B = \beta\sqrt{n}$. There are two cases to consider:
  - If $\tilde{\mathbf{x}} = \perp$, then Sim samples $\mathbf{t}_0, \mathbf{t}_1 \leftarrow_\$ \mathbb{Z}_q^\ell$ and $\mathbf{C} \leftarrow_\$ \mathbb{Z}_q^{k-1\times\ell}$. It sends $\mathsf{ole}_2 = (\mathbf{C}, \mathbf{t}_0, \mathbf{t}_1)$.
  - Else if $\tilde{\mathbf{x}} \neq \perp$, then Sim sets $x = x_i$ where x_i is the i-th coordinate of $\tilde{\mathbf{x}}$. It sends x to $\mathcal{F}_{\mathsf{OLE}}$. When it receives a $\mathbf{y}$ from $\mathcal{F}_{\mathsf{OLE}}$, Sim chooses uniformly at random two vectors $\mathbf{z}_0, \mathbf{z}_1 \in \mathbb{Z}_q^{\ell'}$ such that $\mathbf{z}_0 + x\mathbf{z}_1 = y$. Then, it samples a uniform matrix $\mathbf{U} \leftarrow_\$ \mathbb{Z}_q^{k\times\ell}$ and a matrix $\mathbf{R} \in \mathbb{Z}_q^{n\times\ell}$ where each column $\mathbf{r}^{(j)} \leftarrow_\$ D_{\mathbb{Z}^n,\delta}$ for $j \in [\ell]$ and sets

$$\begin{aligned}\mathbf{C} &= \mathbf{U}_{-i}\\ \mathbf{t}_0 &= \alpha\tilde{\mathbf{x}}\mathbf{U} + \alpha\mathbf{e}\mathbf{R} + \mathsf{ECC.Encode}(\mathbf{z}_0)\\ \mathbf{t}_1 &= -\mathbf{u}_i + \mathsf{ECC.Encode}(\mathbf{z}_1)\end{aligned}$$

    where $\mathbf{u}_i$ is the i-th row of $\mathbf{U}$. It sends $\mathsf{ole}_2 = (\mathbf{C}, \mathbf{t}_0, \mathbf{t}_1)$.

We now proceed to show that the real-world and the ideal-world executions are indistinguishable. The following lemma shows that the CRS generated in the simulation is indistinguishable from one generated in the real-world execution. Then, the next two lemmas deal with the two possible cases in the simulation.

Lemma 8. *The CRS generated above is statistically indistinguishable from a CRS generated according to* GenCRS.

Proof. The only thing that differs in both CRS's is that the matrix $\mathbf{A}$ is generated via TdGen in the simulation (instead of being chosen uniformly). By Lemma 4, it follows that the CRS is statistically indistinguishable from one generated using GenCRS. □

Lemma 9. *Assume that $\tilde{\mathbf{x}} = \perp$. Then, the simulated execution is indistinguishable from the real-world execution.*

Proof. We prove that no (computationally unbounded) adversary can distinguish both executions, except with negligible probability. First, note that, if $\perp = \tilde{\mathbf{x}} \leftarrow \mathsf{Invert}(\mathsf{td}_{\mathbf{A}}, \mathbf{a}')$, then $\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e}$ where $\|\mathbf{e}\| > \beta\sqrt{n}$ since only in this case algorithm Invert fails to *invert* $\mathbf{a}'$.

Consider the matrix $\mathbf{A}' = \begin{pmatrix} \mathbf{A} \\ \mathbf{a}' \end{pmatrix}$. If $\mathbf{a}'$ is of the form described above, then the matrix $\mathbf{A}'$ has no *short* vectors in its row-span. In other words, there is no vector $\mathbf{v} \neq 0$ in the row-span of $\mathbf{A}'$ such that $\|\mathbf{v}\| \leq \beta\sqrt{n}$. This is a direct consequence of the definition of algorithm $\mathsf{InvertCloseVector}$ of Lemma 7.

Hence $\rho_\beta(\Lambda_q(\mathbf{A}') \setminus \{0\}) \leq \mathsf{negl}(\lambda)$. Moreover, we have that

$$\begin{aligned}
\rho_\beta(\Lambda_q(\mathbf{A}') \setminus \{0\}) &\geq \rho_{1/\beta}(\Lambda_q(\mathbf{A}') \setminus \{0\}) \\
&\geq \rho_{1/\delta}(\Lambda_q(\mathbf{A}') \setminus \{0\}) \\
&\geq \rho_{1/(q\delta)}(\Lambda_q(\mathbf{A}') \setminus \{0\}) \\
&= \rho_{1/\delta}(q\Lambda_q(\mathbf{A}') \setminus \{0\}) \\
&= \rho_{1/\delta}((\Lambda_q^\perp(\mathbf{A}'))^* \setminus \{0\})
\end{aligned}$$

where the first and the second inequalities hold because $\delta > \beta > 1$ by hypothesis and the last equality holds because $\frac{1}{q}\Lambda_q^\perp(\mathbf{A}') = \Lambda_q(\mathbf{A}')^*$. Since

$$\rho_{1/\delta}((\Lambda_q^\perp(\mathbf{A}'))^* \setminus \{0\}) \leq \mathsf{negl}(\lambda)$$

then $\delta \geq \eta_\varepsilon(\Lambda^\perp(\mathbf{A}'))$, for $\varepsilon = \mathsf{negl}(\lambda)$, and the conditions of Lemma 2 are met.

Therefore, we can switch to a hybrid experiment where $\mathbf{A}'\mathbf{R} \mod q$ is replaced by $\mathbf{U} \leftarrow_\$ \mathbb{Z}^{(k+1)\times\ell}$ incurring only negligible statistical distance. That is,

$$\begin{pmatrix} \mathbf{C} \\ \mathbf{t}_1 \\ \mathbf{t}_0 \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{-i} \\ \mathbf{a}_i \\ \mathbf{a}' \end{pmatrix} \mathbf{R} + \begin{pmatrix} 0 \\ \hat{\mathbf{z}}_1 \\ \hat{\mathbf{z}}_0 + \tilde{\mathbf{e}} \end{pmatrix} \approx_{\mathsf{negl}(\lambda)} \mathbf{U} + \begin{pmatrix} 0 \\ \hat{\mathbf{z}}_1 \\ \hat{\mathbf{z}}_0 + \tilde{\mathbf{e}} \end{pmatrix} \approx_{\mathsf{negl}(\lambda)} \mathbf{U}$$

where $\hat{\mathbf{z}}_j$ is the encoding of $\mathsf{ECC.Encode}(\mathbf{z}_j)$ for $j \in \{0, 1\}$.

We conclude that, in this case, the real-world and the ideal-world execution (where Sim just sends a uniformly chosen triple $(\mathbf{C}, \mathbf{t}_0, \mathbf{t}_1)$) are statistically indistinguishable. □

Lemma 10. *Assume that $\tilde{\mathbf{x}} \neq \perp$. Then, the simulated execution is indistinguishable from the real-world execution.*

Proof. The proof follows the following sequence of hybrids:

Hybrid $\mathcal{H}_0$. This is the real-world protocol. In particular, in this hybrid, the simulator behaves as the honest sender and computes

$$\begin{aligned}
&\mathbf{t}_0 = \mathbf{a}'\mathbf{R} + \mathsf{ECC.Encode}(\mathbf{z}_0) = \alpha\tilde{\mathbf{x}}\mathbf{A}\mathbf{R} + \alpha\mathbf{e}\mathbf{R} + \mathsf{ECC.Encode}(\mathbf{z}_0) \mod q\\
&\mathbf{t}_1 = \mathbf{a}_i\mathbf{R} + \mathsf{ECC.Encode}(\mathbf{z}_1) \mod q\\
&\mathbf{C} = \mathbf{A}_{-i}\mathbf{R} \mod q
\end{aligned}$$

for some $\alpha \in \mathbb{Z}_q \setminus \{0\}$.

Hybrid $\mathcal{H}_1$. This hybrid is similar to the previous one, except that Sim computes $\mathbf{t}_0$ as $\alpha\mathbf{x}\mathbf{U} + \alpha\mathbf{e}\mathbf{R} + \mathsf{ECC.Encode}(\mathbf{z}_0)$, $\mathbf{C}$ by $\mathbf{U}_{-i}$ and $\mathbf{t}_1$ by $\mathbf{u}_i + \mathsf{ECC.Encode}(\mathbf{z}_1)$, where $\mathbf{u}_i$ is the i-th row of $\mathbf{U} \leftarrow_\$ \mathbb{Z}_q^{k\times\ell}$.

This hybrid corresponds to the simulator for the corrupted receiver.

Claim 1. $|\Pr[1 \leftarrow \mathcal{A} : \mathcal{A} \text{ plays } \mathcal{H}_0] - \Pr[1 \leftarrow \mathcal{A} : \mathcal{A} \text{ plays } \mathcal{H}_1]| \leq \mathsf{negl}(\lambda)$.

To prove this claim, we will resort to the partial smoothing lemma (Lemma 3). Using the same notation as in Lemma 3, consider $\gamma = \beta\sqrt{n}$. Then, we have that

$$\mathsf{negl}(\lambda) \geq \rho_\beta(\Lambda_q(\mathbf{A}') \setminus \gamma\mathcal{B}) \geq \rho_{q/\delta}(\Lambda_q(\mathbf{A}') \setminus \gamma\mathcal{B})$$

since, by assumption, $\beta > q/\delta$.

Hence, by applying Lemma 3, we obtain

$$\mathbf{A}\mathbf{R} \mod q \approx_{\mathsf{negl}(\lambda)} \mathbf{A}(\mathbf{R} + \mathbf{X}) \mod q$$

for $\mathbf{X} \leftarrow_\$ \Lambda^\perp(\mathbf{e})$ (here, in the notation of Lemma 3, we consider $\mathbf{D} = \mathbf{e}$).

We now argue that $\mathbf{A}\mathbf{X} \mod q \approx_{\mathsf{negl}(\lambda)} \mathbf{U}$ for $\mathbf{U} \leftarrow_\$ \mathbb{Z}_q^{k\times\ell}$. Let $\mathbf{B} \in \mathbb{Z}_q^{n\times k'}$ be a basis of $\Lambda^\perp(\mathbf{e})$, that is, $\mathbf{e}\mathbf{B} = 0$. Let us assume for the sake of contradiction that $\mathbf{A}\mathbf{B}$ does not have full rank (hence, $\mathbf{A}\mathbf{X} \mod q$ is not uniform over $\mathbb{Z}_q^{k\times\ell}$). Then, there is a vector $\mathbf{v} \in \mathbb{Z}_q^k$ such that $\mathbf{v}\mathbf{A}\mathbf{B} = 0$.

Since $\mathbf{B}$ is a basis of $\Lambda^\perp(\mathbf{e})$, this means that $\mathbf{v}\mathbf{B} \in (\Lambda^\perp(\mathbf{e}))^\perp = \Lambda(\mathbf{e})$. In other words, $\mathbf{v}\mathbf{A} = t \cdot \mathbf{e}$ for some $t \in \mathbb{Z}_q$. Consequently, we have $\mathbf{e} = t^{-1}\mathbf{v}\mathbf{A}$ and thus $\mathbf{e}$ is in the row-span of $\mathbf{A}$, that is, $\mathbf{A}$ has a vector of norm shorter than $\beta\sqrt{n}$ in its row-span. However, this happens only with negligible probability over the uniform choice of $\mathbf{A}$ and, thus, we reach a contradiction. We conclude that $\mathbf{A}\mathbf{B}$ needs to have full rank. Now, since $\mathbf{X}$ is sampled uniformly from $\Lambda^\perp(\mathbf{e})$, we have that $\mathbf{A}\mathbf{X}$ is uniform over $\mathbb{Z}_q^{k\times\ell}$. Thus, $\mathbf{A}\mathbf{X} \mod q \approx_{\mathsf{negl}(\lambda)} \mathbf{U}$ where $\mathbf{U} \leftarrow_\$ \mathbb{Z}_q^{k\times\ell}$.

Claim 2. *Let* R *be any receiver. The values* $(\mathbf{z}_0, \mathbf{z}_1)$ *are perfectly hidden from* R *given that* $\mathbf{z}_0 + x\mathbf{x}_1 = \mathbf{y}$.

To see this, consider again the values computed by Sim in this hybrid

$$\begin{aligned}
&\mathbf{C} = \mathbf{U}_{-i}\\
&\mathbf{t}_0 = \alpha\tilde{\mathbf{x}}\mathbf{U} + \alpha\tilde{\mathbf{e}} + \mathsf{ECC.Encode}(\mathbf{z}_0)\\
&\mathbf{t}_1 = -\mathbf{u}_i + \mathsf{ECC.Encode}(\mathbf{z}_1)
\end{aligned}$$

where $\tilde{\mathbf{e}} = \alpha \mathbf{e}\mathbf{R}$.

There are two cases to consider: either $x_i = 0$ or $x_i \neq 0$. In the first case, neither $\mathbf{C}$ nor $\mathbf{t}_0$ carry information about $\mathbf{u}_i$, which is a uniformly chosen vector. Hence, $\mathbf{z}_1$ is perfectly hidden from R.

When $x_i \neq 0$, consider another pair $(\mathbf{z}'_0, \mathbf{z}'_1)$ such that $\mathbf{z}_0 + x_i\mathbf{z}_1 = \mathbf{z}'_0 + x_i\mathbf{z}'_1$. Then, the probability that Sim runs the protocol on input $(\mathbf{z}_0, \mathbf{z}_1)$ or $(\mathbf{z}'_0, \mathbf{z}'_1)$ is exactly the same from the point-of-view of R, given that $\mathbf{U} \leftarrow_\$ \mathbb{Z}_q^{k\times\ell}$. To see this, note that given $\mathbf{C}, \mathbf{t}_0, \mathbf{t}_1$, we can set $\mathbf{u}_i = \mathbf{s}_1 - \mathsf{ECC.Encode}(\mathbf{z}'_1) = \alpha x_i^{-1}(\mathbf{s}_0 - \mathsf{ECC.Encode}(\mathbf{z}'_0))$, which is uniform in $\mathbb{Z}_q$ since q is prime. A simple calculation shows that correctness still holds in this case. □

This concludes the description of the simulator for the corrupted receiver. We now resume the proof of Theorem 2 by presenting the simulator for the semi-honest sender.

Simulator for corrupted sender. We describe how the simulator Sim proceeds: It takes S's inputs $(\mathbf{z}_0, \mathbf{z}_1)$ and sends them to the ideal functionality $\mathcal{F}_{\mathsf{OLE}}$, which returns nothing. It simulates the dummy R by sampling $\mathbf{a}' \leftarrow_\$ \mathbb{Z}_q^n$ and sending it to the corrupted sender.

It is trivial to see that both the ideal and the real-world executions are indistinguishable given that the $\mathsf{LWE}_{k,q,\beta}$ assumption holds. □

5.3 Batch OLE

We now show how we can extend the protocol described above in order to implement a batch reusable OLE protocol, that is, in order to implement the functionality $\mathcal{F}_{\mathsf{bOLE}}$ described in Section 3.

This variant improves the efficiency of the protocol since the receiver R can *commit* to a batch of inputs $\{x_i\}_{i\in[k']}$, and not just one input, using the same first message of the two-round OLE. Hence, the size of the first message can be amortized over the number of R's inputs, to achieve a better communication complexity.

Construction 4. *The protocol is composed by the algorithms* $(\mathsf{GenCRS}, \mathsf{R}_1, \mathsf{S}, \mathsf{R}_2)$. *Let* $k, n, \ell, \ell', q, k' \in \mathbb{Z}$ *such that* q *is a prime and* $n = \mathsf{poly}(k \log q)$, *and let* $\beta, \delta, \xi \in \mathbb{R}$ *such that* $\frac{q}{\sqrt{n}\tau(k)} > \beta$ *(where* $\tau(k) = \omega(\sqrt{\log k})$ *as in Lemma 4),* $\delta > \beta > 1$ *and* $\beta > q/\delta$.

$\mathsf{GenCRS}(1^\lambda)$**:** *This algorithm is similar to the one described in Construction 3.*

$\mathsf{R}_1\left(\mathsf{crs}, \{x_j\}_{j\in[k']} \in \mathbb{Z}_q\right)$**:** *The algorithm is similar to the one described in Construction 3, except that it outputs* $\mathsf{ole}_1 = (\mathbf{a}', k')$ *and* $\mathsf{st} = \mathbf{x}$, *where* $\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e}$ *and* $(x_1, \ldots, x_{k'})$ *corresponds to the first* k' *of* $\mathbf{x}$.

$\mathsf{S}\left(\mathsf{crs}, (\mathbf{z}_0, \mathbf{z}_1) \in (\mathbb{Z}_q^{\ell'})^2, m_{\mathsf{R}}, j \in [k']\right)$**:** *This algorithm is similar to the one described in Construction 3, except that; i) it additionally uses j as the index i; ii) it outputs j (which corresponds to which x_j the receiver* R *is supposed to use in that particular execution of the protocol).*

$\mathsf{R}_2(\mathsf{crs}, \mathsf{st}, m_{\mathsf{S}})$**:** *This algorithm is similar to the one described in Construction 3, except that it outputs $\mathbf{z}_0 + x_j\mathbf{z}_1 = \mathbf{y} \leftarrow \mathsf{ECC.Decode}(\mathbf{x}_{-j}\mathbf{C} - (\mathbf{t}_0 + x_j\mathbf{t}_1))$.*

It is clear that correctness still holds.

Theorem 3 (Security)**.** *Assume that the $\mathsf{LWE}_{k,\beta,q}$ assumption holds, $\frac{q}{\sqrt{n}\tau(k)} > \beta$ (where $\tau(k) = \omega(\sqrt{\log k})$ as in Lemma 4), $\delta > \beta > 1$ and $\beta > q/\delta$. The protocol presented in Construction 4 securely realizes the functionality $\mathcal{F}_{\mathsf{bOLE}}$ in the $\mathcal{G}_{\mathsf{CRS}}$-hybrid model against:*

- *a semi-honest sender given that the $\mathsf{LWE}_{k-k',\beta,q}$ assumption holds;*
- *a malicious receiver where security holds statistically.*

The proof of the theorem stated above essentially follows the same blueprint as the proof of Theorem 2, except that the simulator for the corrupted receiver extracts the first k' coordinates $\{x_j\}_{j\in[k']}$ of $\mathbf{x}$ and sends these values to $\mathcal{F}_{bOLE}$. From now on, it behaves exactly as the simulator in the proof of Theorem 2. Indistinguishability of executions follows exactly the same reasoning.

6 OLE from LWE secure against Malicious Adversaries

In this section, we extend the construction of the previous section to support malicious sender. The idea is to use a *cut-and-choose* approach via the use of an OT scheme in two rounds and extract the sender's input via the OT simulator.

6.1 Protocol

Construction 5. *The protocol is composed by the algorithms* $(\mathsf{GenCRS}, \mathsf{R}_1, \mathsf{S}, \mathsf{R}_2)$*. Let $k, n, \ell, \ell', q \in \mathbb{Z}$ such that q is a prime and $n = \mathsf{poly}(k \log q)$, and let $\beta, \delta, \xi \in \mathbb{R}$ such that $\frac{q}{\sqrt{n}\tau(k)} > \beta$ (where $\tau(k) = \omega(\sqrt{\log k})$ as in Lemma 4), $\delta > \beta > 1$ and $\beta > q/\delta$. $\mathcal{F}_{\mathsf{OT}}$ is the OT functionality as in Section 3. We now present the protocol in full detail.*

$\mathsf{GenCRS}(1^\lambda)$**:**

- *Sample* $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{k\times n}$.
- *Choose a linear ECC* $\mathsf{ECC}_{\ell',\ell,\xi} = (\mathsf{ECC.Encode}, \mathsf{ECC.Decode})$ *over* $\mathbb{Z}_q$.
- *Output* $\mathsf{crs} = (\mathbf{A}, \mathsf{ECC}_{\ell',\ell,\xi})$.

$\mathsf{R}_1\left(\mathsf{crs}, x \in \mathbb{Z}_q\right)$**:**

- *Parse* crs *as* $(\mathbf{A}, \mathsf{ECC}_{\ell',\ell,\xi})$.
- *Sample* $\mathbf{x} = (x_1, \ldots, x_k) \leftarrow_\$ \mathbb{Z}_q^k$ *such that* $x_i = x$ *and a small error vector* $\mathbf{e} \leftarrow_\$ \chi_\beta^n$, *for some index* $i \leftarrow_\$ [k]$.
- *Compute* $\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e}$.
- *Additionally choose uniformly at random* $\mathbf{b} = (b_1, \ldots, b_\lambda) \leftarrow_\$ \{0,1\}^\lambda$ *such that the weight* $wt(\mathbf{b}) = \lambda/2$ *(i.e., half of the coordinates are* 0*). Send* $\{b_j\}_{j\in[\lambda]}$ *to the OT functionality* $\mathcal{F}_{\mathsf{OT}}$.
- *Output* $\mathsf{ole}_1 = (\mathbf{a}', i)$ *and* $\mathsf{st} = \big(\mathbf{x}, i, \{b_j\}_{j\in[\lambda]}\big)$.

$\mathsf{S}\left(\mathsf{crs}, (\mathbf{z}_0, \mathbf{z}_1) \in \mathbb{Z}_q^{\ell'}, \mathsf{ole}_1\right)$**:**

- *Parse* crs *as* $(\mathbf{A}, \mathsf{ECC}_{\ell',\ell,\xi})$ *and* ole_1 *as* $(\mathbf{a}', i)$.
- *For all* $j \in [\lambda]$, *do the following:*
  - *Sample* $\mathbf{R}_j \in \mathbb{Z}_q^{n\times\ell}$ *where each column is sampled according to* $D_{\mathbb{Z}^n,\delta}$.
  - *Compute* $\mathbf{s}_{0,j}, \mathbf{s}_{1,j} \in \mathbb{Z}_q^\ell$ *such that* $\mathbf{s}_{0,j} = \mathbf{a}'\mathbf{R}_j$, $\mathbf{s}_{1,j} = \mathbf{a}_i\mathbf{R}_j$, *and* $\mathbf{C}_j = \mathbf{A}_{-i}\mathbf{R}_j \in \mathbb{Z}_q^{(k-1)\times\ell}$.
  - *Compute* $\mathbf{t}_{0,j} = \mathbf{s}_{0,j} + \mathsf{ECC.Encode}(\mathbf{u}_{0,j})$ *and* $\mathbf{t}_{1,j} = \mathbf{s}_{1,j} + \mathsf{ECC.Encode}(\mathbf{u}_{1,j})$, *for uniformly chosen* $\mathbf{u}_{0,j}, \mathbf{u}_{1,j} \leftarrow_\$ \mathbb{Z}_q^\ell$.
  - *Send* $M_{0,j}, M_{1,j}$ *to* $\mathcal{F}_{\mathsf{OT}}$ *where* $M_{0,j} = \mathbf{R}_j$ *and* $M_{1,j} = (\mathbf{t}_{0,j}, \mathbf{t}_{1,j}, \mathbf{u}_{0,j} + \mathbf{z}_0, \mathbf{u}_{1,j} + \mathbf{z}_1)$.
- *Output* $\mathsf{ole}_2 = \{\mathbf{C}_j\}_{j\in[\lambda]}$.

$\mathsf{R}_2(\mathsf{crs}, \mathsf{st}, \mathsf{ole}_2)$**:**

- *Parse* crs *as* $(\mathbf{A}, \mathsf{ECC}_{\ell',\ell,\xi})$, ole_2 *as* $\{\mathbf{C}_j\}_{j\in[\lambda]}$ *and* st *as* $\big(\mathbf{x}, i, \{b_j\}_{j\in[\lambda]}\big)$.
- *If any of the matrices* $\mathbf{C}_j$ *is not full-rank, abort the protocol.*
- *For all* $j \in [\lambda]$, *do the following:*
  - *Recover* $M_{b_j,j}$ *from* $\mathcal{F}_{\mathsf{OT}}$.
  - *If* $b_j = 0$, *parse* $M_{0,j} = \tilde{\mathbf{R}}_j$. *If* $\mathbf{C}_j \neq \mathbf{A}_{-i}\tilde{\mathbf{R}}_j$ *and* $\tilde{\mathbf{R}}_j$ *is not small, abort the protocol.*
  - *If* $b_j = 1$, *parse* $M_{1,j}$ *as* $(\tilde{\mathbf{t}}_{0,j}, \tilde{\mathbf{t}}_{1,j}, \tilde{\mathbf{v}}_{0,j}, \tilde{\mathbf{v}}_{1,j})$. *Compute* $\tilde{\mathbf{y}}_j = \mathbf{x}_{-i}\mathbf{C}_j - (\tilde{\mathbf{t}}_{0,j} + x_i\tilde{\mathbf{t}}_{1,j})$ *and then run* $\mathbf{y}_j \leftarrow \mathsf{ECC.Decode}(\tilde{\mathbf{y}}_j)$. *If* $\mathbf{y}_j = \perp$, *abort the protocol. Else, compute* $\mathbf{w}_j = \tilde{\mathbf{v}}_{0,j} + x_i\tilde{\mathbf{v}}_{1,j} - \mathbf{y}_j$.
- *Check if* $\mathbf{w}_j = \mathbf{w}_{j'}$ *for all* (j, j') *such that* $b_j = b_{j'} = 1$. *If the test fails, abort the protocol.*
- *For any* j *such that* $b_j = 1$, *set* $\mathbf{w} = \mathbf{w}_j$. *Output* $\mathbf{w} \in \mathbb{Z}_q^{\ell'}$.

6.2 Analysis

We now proceed to the analysis of the protocol described above.

Theorem 4 (Correctness)**.** *Let $\mathsf{ECC}_{\ell',\ell,\xi}$ be a linear ECC where $\xi \geq \sqrt{\ell}\beta\delta n$. Then the protocol presented in Construction 5 is correct.*

Proof. From the proof of Theorem 1, we have that $\mathbf{y}_j = \mathbf{u}_{0,j} + x_i \mathbf{u}_{1,j}$, except with negligible probability, when $b_j = 1$. Hence $\mathbf{w}_j = \tilde{\mathbf{v}}_{0,j} + x_i \tilde{\mathbf{v}}_{1,j} - \mathbf{y}_j = \mathbf{z}_0 + x_i \mathbf{z}_1$, which is equal to every other $\mathbf{w}_{j'}$ when $b_j = b_{j'} = 1$. □

Theorem 5 (Security)**.** *Assume that the $\mathsf{LWE}_{k,\beta,q}$ assumption holds, $\frac{q}{\sqrt{n}\tau(k)} > \beta$ (where $\tau(k) = \omega(\sqrt{\log k})$ as in Lemma 4), $\delta > \beta > 1$ and $\beta > q/\delta$. The protocol presented in Construction 5 securely realizes the functionality $\mathcal{F}_{\mathsf{OLE}}$ in the $(\mathcal{G}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{OT}})$-hybrid model against static malicious adversaries where:*

- *Security against corrupted sender holds given that the $\mathsf{LWE}_{k,\beta,q}$ assumption holds.*
- *Security against a corrupted receiver holds statistically in the $(\mathcal{G}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{OT}})$-hybrid model.*

Proof. In order to prove security against a static malicious receiver we have to show that there is a simulator that can simulate the view of the adversary in the ideal world.

Simulator for corrupted receiver: The proof is essentially the same as the proof of Theorem 2. Still, we present the simulator.

Let $(\mathsf{TdGen}, \mathsf{Invert})$ be the algorithms described in Lemma 4 and $\mathsf{InvertCloseVector}$ the algorithm of Lemma 7.

- **CRS generation:** Sim behaves as the simulator of the proof of Theorem 2 and additionally simulates $\mathcal{F}_{\mathsf{OT}}$.
- Upon receiving a message $\mathbf{a}'$ from R and $\{b_j\}_{j\in[\lambda]}$ through $\mathcal{F}_{\mathsf{OT}}$, Sim runs $(\tilde{\mathbf{x}}, \alpha, \mathbf{e}) \leftarrow \mathsf{InvertCloseVector}(\mathsf{td}_{\mathbf{A}}, \mathbf{a}')$. There are two cases to consider:
  - If $\tilde{\mathbf{x}} = \perp$, then for each $j \in [\lambda]$ Sim does the following:
    - ∗ If $b_j = 0$, then Sim computes $\mathbf{C}_j = \mathbf{A}_{-i}\mathbf{R}_j$. It sets $M_{0,j} = \mathbf{R}_j$ and $M_{1,j} \leftarrow_{\$} \mathbb{Z}_q^{\ell} \times \mathbb{Z}_q^{\ell} \times \mathbb{Z}_q^{\ell'} \times \mathbb{Z}_q^{\ell'}$.
    - ∗ If $b_j = 1$, then Sim sets $\mathbf{C}_j \leftarrow_{\$} \mathbb{Z}_q^{k-1\times\ell}$, $M_{0,j} \leftarrow_{\$} \mathbb{Z}_q^{n\times\ell}$ and $M_{1,j} \leftarrow_{\$} \mathbb{Z}_q^{\ell} \times \mathbb{Z}_q^{\ell} \times \mathbb{Z}_q^{\ell'} \times \mathbb{Z}_q^{\ell'}$.
  - Else if $\tilde{\mathbf{x}} \neq \perp$, then Sim sets $x = x_i$ where x_i is the i-th coordinate of $\tilde{\mathbf{x}}$. It sends x to $\mathcal{F}_{\mathsf{OLE}}$. Upon receiving a $\mathbf{y}$ from $\mathcal{F}_{\mathsf{OLE}}$, Sim chooses uniformly at random two vectors $\mathbf{z}_0, \mathbf{z}_1 \in \mathbb{Z}_q^{\ell'}$ such that $\mathbf{z}_0 + x\mathbf{z}_1 = y$. Then, for every $j \in [\lambda]$, it does the following:
    - ∗ If $b_j = 0$, then Sim computes $\mathbf{C}_j = \mathbf{A}_{-i}\mathbf{R}_j$. It sets $M_{0,j} = \mathbf{R}_j$ and $M_{1,j} \leftarrow_{\$} \mathbb{Z}_q^{\ell} \times \mathbb{Z}_q^{\ell} \times \mathbb{Z}_q^{\ell'} \times \mathbb{Z}_q^{\ell'}$.

* If $b_j = 1$, then Sim samples a uniform matrix $\mathbf{V}_j \leftarrow_\$ \mathbb{Z}_q^{k\times\ell}$ and a matrix $\mathbf{R}_j \in \mathbb{Z}_q^{n\times\ell}$ where each column $\mathbf{r}^{(\kappa)} \leftarrow_\$ D_{\mathbb{Z}^n,\delta}$ for $\kappa \in [\ell]$ and sets

$$\begin{aligned}\mathbf{C}_j &= \mathbf{V}_{-i,j}\\ \mathbf{t}_0 &= \alpha\tilde{\mathbf{x}}\mathbf{V}_j + \alpha\mathbf{e}\mathbf{R}_j + \mathsf{ECC.Encode}(\mathbf{u}_{0,j})\\ \mathbf{t}_1 &= -\mathbf{v}_{i,j} + \mathsf{ECC.Encode}(\mathbf{u}_{1,j})\end{aligned}$$

  where $\mathbf{v}_{i,j}$ is the i-th row of $\mathbf{V}_j$, and $\mathbf{V}_{-i,j}$ is the matrix $\mathbf{V}_j$ with the i-th row removed. It sets $M_{0,j} \leftarrow_\$ \mathbb{Z}_q^{n\times\ell}$ and $M_{1,j} = (\mathbf{t}_0, \mathbf{t}_1, \mathbf{u}_{0,j} + \mathbf{z}_0, \mathbf{u}_{1,j} + \mathbf{z}_1)$.
* It sends $\mathsf{ole}_2 = \{\mathbf{C}_j\}$.

The proof of indistinguishability of executions is essentially the same as the proof in Theorem 2. We stress that security still holds statistically in the $\mathcal{F}_{\mathsf{OT}}$ model.

Simulator for corrupted sender: We describe the simulator Sim against a corrupted sender.

- **CRS generation:** Sim generates crs as in GenCRS.
- Sim computes $\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e} \in \mathbb{Z}_q^n$ for a uniformly chosen $\mathbf{x} \leftarrow_\$ \mathbb{Z}_q^k$ and $\mathbf{e} \leftarrow_\$ \mathbb{D}_{\mathbb{Z}^n,\beta}$, $i \leftarrow_\$ [k]$ and sends $(i, \mathbf{a}')$ to the sender.
- Upon receiving $(M_{0,j}, M_{1,j})$ from the sender (a message intended to $\mathcal{F}_{\mathsf{OT}}$), Sim does the following:
  - It chooses a partition of size $\lambda/2$ of $[\lambda]$. Let us denote this partition by $I_0 \cup I_1 = [\lambda]$.
  - It parses $M_{0,j}$ as $\mathbf{R}_j$ and $M_{1,j}$ as $(\mathbf{t}_{0,j}, \mathbf{t}_{1,j}, \mathbf{v}_{0,j}, \mathbf{v}_{1,j})$.
  - It checks if $\mathbf{C}_j = \mathbf{A}_{-i}\mathbf{R}_j$ for $j \in I_0$. If this does not happen for at least one index $j \in I_0$, it aborts.
  - Similarly, it checks if $\mathbf{w}_j = \mathbf{w}_{j'}$ for any pair of indices $(j, j') \in I_1 \times I_1$, where $\mathbf{w}_j = \mathbf{v}_{0,j} + x_i\mathbf{v}_{1,j} - \mathsf{ECC.Decode}(\mathbf{x}_{-i}\mathbf{C}_j - (\mathbf{t}_{0,j} + x_i\mathbf{t}_{1,j}))$. If this does not happen for at least one pair of indices, it aborts.
  - Check if there are positions j for which both $M_{0,j}$ and $M_{1,j}$ values are *correct*, meaning that the honest receiver does not abort for any of these values. Let j' be one of these positions. Then, Sim extracts $\mathbf{z}_0, \mathbf{z}_1$ in the following way: It computes $\mathbf{s}_{0,j}$, which allows to compute $\mathbf{u}_{0,j}$. Finally, it recovers $\mathbf{z}_0$ from $\mathbf{u}_{0,j} + \mathbf{z}_0$. The value $\mathbf{z}_1$ can be recovered similarly. It sends $\mathbf{z}_0, \mathbf{z}_1$ to $\mathcal{F}_{\mathsf{OLE}}$.

We first show that the Sim described above is correct and efficient.

Lemma 11. Sim *succeeds in extracting a pair* $(\mathbf{z}_0, \mathbf{z}_1)$*, except with negligible probability.*

Proof. The probability of Sim not extracting any pair $(\mathbf{z}_0, \mathbf{z}_1)$ is equal to the probability of not existing any position j' such that both values in the OT are correct given that Sim has not aborted before.

Note that this case only happens if the corrupted sender inputs malformed values for $M_{0,j}$ in $\lambda/2$ positions and malformed values for $M_{1,j}$ in the remaining $\lambda/2$ positions. Let E be the event where Sim aborts given this scenario. This happens with probability

$$\Pr[E] = 1 - \Pr[X = 0]$$

where $X \leftarrow\$ \, \mathit{HyperGeom}(\lambda/2, \lambda/2, \lambda)$ where *HyperGeom* is a hypergeometric distribution with parameters $(\lambda/2, \lambda/2, \lambda)$. Thus,

$$\Pr[E] = 1 - \frac{\binom{\lambda/2}{0}\binom{\lambda/2}{\lambda/2}}{\binom{\lambda}{\lambda/2}} = 1 - \mathsf{negl}(\lambda)\,.$$

We conclude that Sim aborts, except with negligible probability. Thus, it extracts a pair $(\mathbf{z}_0, \mathbf{z}_1)$, except with negligible probability. □

Lemma 12. *The extracted pair* $(\mathbf{z}_0, \mathbf{z}_1)$ *is unique, given that the* $\mathsf{LWE}_{k,n,\beta,q}$ *assumption holds.*

Proof. Assume that there is an adversary $\mathcal{A}$ corrupting the sender that is able to embed two pairs $(\mathbf{z}_0, \mathbf{z}_1) \neq (\mathbf{z}'_0, \mathbf{z}'_1)$ such that $\mathbf{z}_0 + x_i\mathbf{z}_1 = \mathbf{z}'_0 + x_i\mathbf{z}'_1$ in the execution of the protocol, with non-negligible probability ε. We show that, if this happens, then we can build an algorithm $\mathcal{B}$ that attacks the $\mathsf{LWE}_{k,\beta,q}$. The reduction works by running several copies of $\mathcal{A}$ simultaneously in parallel and by checking if the outputs of $\mathcal{A}$ in each of these executions are consistent with one another or if they are completely independent.

The algorithm $\mathcal{B}$ takes as input an LWE challenge $(\mathbf{A}, \mathbf{a}') \in \mathbb{Z}_q^{k\times(n\lambda/\varepsilon)}$, parses $\mathbf{A} = (\mathbf{A}_1|\dots|\mathbf{A}_{\lambda/\varepsilon})$ and $\mathbf{a}' = (\mathbf{a}'_1|\dots|\mathbf{a}'_{\lambda/\varepsilon})$. Now, $\mathcal{B}$ runs λ/ε executions of $\mathcal{A}$ in parallel where, at each execution $j \in [\lambda/\varepsilon]$, it embeds $\mathbf{A}_j$ in the crs and simulate the receiver in the protocol by sending $(\mathbf{a}'_j, i)$, for $i \leftarrow\$ \, [k]$ (the index i is the same across all executions j). Then, $\mathcal{B}$ extracts two different pairs $(\mathbf{z}_0, \mathbf{z}_1) \neq (\mathbf{z}'_0, \mathbf{z}'_1)$ while simulating $\mathcal{F}_{\mathsf{OT}}$ and computes for which x_i we have $\mathbf{z}_0 + x_i\mathbf{z}_1 = \mathbf{z}'_0 + x_i\mathbf{z}'_1$. If the extracted value x_i is the same in $d \approx \lambda$ executions, then $\mathcal{B}$ outputs 1. Else, it outputs 0.

We now analyze success probability of $\mathcal{B}$. If

$$\mathbf{a}' = \mathbf{x}\mathbf{A} + \mathbf{e} = \big(\mathbf{x}\mathbf{A}_1 + \mathbf{e}_1|\dots|\mathbf{x}\mathbf{A}_{\lambda/\varepsilon} + \mathbf{e}_{\lambda/\varepsilon}\big)$$

for some error vectors $\mathbf{e}_1, \dots, \mathbf{e}_{\lambda/\varepsilon}$, then after repeating this process λ/ε, we get that the extracted value x_i is expected to be the same in λ executions. In this case, $\mathcal{B}$ outputs 1. Else if $\mathbf{a}' \leftarrow\$ \, \mathbb{Z}_q^n$, then the vector $\mathbf{x}$ (and thus the extracted x_i) is independent of $\mathbf{a}'$. Hence the probability of extracting the same x_i in λ/ε executions is $|\mathbb{Z}_q|^{-\lambda/\varepsilon} = \mathsf{negl}(\lambda)$. In this case, $\mathcal{B}$ outputs 0 guessing that $\mathbf{a}'$ is a uniform vector.

Hence, the advantage of $\mathcal{B}$ in attacking the $\mathsf{LWE}_{k,\beta,q}$ is non-negligible. □

Lemma 13. *The ideal-world and real-world executions are indistinguishable, given that the* $\mathsf{LWE}_{k,\beta,q}$ *assumption holds.*

Proof. The simulator behaves and aborts with exactly the same probability as the real-world receiver. □

The theorem follows from the lemmas above. □

6.3 Instantiating the Functionality $\mathcal{F}_{\mathsf{OT}}$

Several two-round OT protocols exist in the literature that are proven to be UC-secure [PVW08, DGH$^+$20]. When we instantiate our protocol with the OT schemes from [PVW08, Qua20] we obtain several nice properties for the OLE scheme, namely:

1. The protocol is still two-round.
2. The protocol preserves statistical security against a corrupted receiver, since the OT of [PVW08, Qua20] is also statistically secure against a corrupted receiver.
3. The OLE scheme is secure based only on the LWE assumption.

Acknowledgment

Pedro Branco thanks the support from DP-PMI and FCT (Portugal) through the grant PD/BD/135181/2017. This work is supported by Security and Quantum Information Group of Instituto de Telecomunicações, by the Fundação para a Ciência e a Tecnologia (FCT) through national funds, by FEDER, COMPETE 2020, and by Regional Operational Program of Lisbon, under UIDB/50008/2020.

References

[ADI$^+$17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 223–254, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[AP12] Jacob Alperin-Sheriff and Chris Peikert. Circular and KDM security for identity-based encryption. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages

334–352, Darmstadt, Germany, May 21–23, 2012. Springer, Heidelberg, Germany.

[Ban93] W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(4):625–636, 1993.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[BD18] Zvika Brakerski and Nico Döttling. Two-message statistically sender-private OT from LWE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 370–390, Panaji, India, November 11–14, 2018. Springer, Heidelberg, Germany.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 500–532, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

[CDI$^+$19] Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. Reusable non-interactive secure computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 462–488, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[DGH$^+$20] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. Two-round oblivious transfer from CDH or LPN. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 768–797, Cham, 2020. Springer International Publishing.

[DGHM18] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. In Michel Abdalla and Ricardo

Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 3–31, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany.

[DGN$^+$17] Nico Döttling, Satrajit Ghosh, Jesper Buus Nielsen, Tobias Nilges, and Roberto Trifiletti. TinyOLE: Efficient actively secure two-party computation from oblivious linear function evaluation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 2263–2276, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[DKM12] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Statistically secure linear-rate dimension extension for oblivious affine function evaluation. In Adam Smith, editor, *ICITS 12: 6th International Conference on Information Theoretic Security*, volume 7412 of *Lecture Notes in Computer Science*, pages 111–128, Montreal, QC, Canada, August 15–17, 2012. Springer, Heidelberg, Germany.

[DKMQ12] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. David & Goliath oblivious affine function evaluation - asymptotically optimal building blocks for universally composable two-party computation from a single untrusted stateful tamper-proof hardware token. Cryptology ePrint Archive, Report 2012/135, 2012. `https://eprint.iacr.org/2012/135`.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[GN19] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 154–185, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[GNN17] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 629–659, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E.

Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, BC, Canada, May 17–20, 2008. ACM Press.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 468–499, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[GS19] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 3–29, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[HIMV19] Carmit Hazay, Yuval Ishai, Antonio Marcedone, and Muthuramakrishnan Venkitasubramaniam. LevioSA: Lightweight secure arithmetic computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 327–344. ACM Press, November 11–15, 2019.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, *Theory of Cryptography*, pages 294–314, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018: 27th USENIX Security Symposium*, pages 1651–1669, Baltimore, MD, USA, August 15–17, 2018. USENIX Association.

[LP94] Mody Lempel and Azaria Paz. An algorithm for finding a shortest vector in a two-dimensional modular lattice. *Theoretical Computer Science*, 125(2):229 – 241, 1994.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007.

[MZ17] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[Qua20] Willy Quach. UC-secure OT from LWE, revisited. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks*, pages 192–211, Cham, 2020. Springer International Publishing.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.