Two-Round Maliciously-Secure Oblivious Transfer with Optimal Rate

Pedro Branco[1], Nico Döttling[2], and Akshayaram Srinivasan[3]

[1]Max Planck Institute for Security and Privacy
[2]Helmholtz Center for Information Security (CISPA)
[3]University of Toronto

Abstract

We give a construction of a two-round batch oblivious transfer (OT) protocol in the CRS model that is UC-secure against malicious adversaries and has (near) optimal communication cost. Specifically, to perform a batch of k oblivious transfers where the sender's inputs are bits, the sender and the receiver need to communicate a total of $3k + o(k) \cdot \mathsf{poly}(\lambda)$ bits. We argue that $3k$ bits are required by any protocol with a black-box and straight-line simulator. The security of our construction is proven assuming the hardness of Quadratic Residuosity (QR) and the Learning Parity with Noise (LPN).

1 Introduction

Oblivious transfer (OT) is a two-party protocol between a sender and a receiver. The receiver's private input in this protocol is a bit $b \in \{0, 1\}$ and the sender's private input are two bits (m_0, m_1). At the end of the protocol, the receiver learns m_b and the sender does not get any output. For security, we require that the sender learns no information about the receiver's choice bit b and the receiver should not learn any information about the sender's other bit, namely, m_{1-b}. We also consider a natural extension where the sender's private inputs are two strings rather than bits and the receiver learns one of the two strings. This extension is known as the string OT protocol.

Oblivious transfer is a foundational cryptographic primitive that is used as the core building block in the construction of secure computation protocols [Yao86, Kil88, IPS08, GS18, BL18]. In many applications, including secure computation and OT extension [Bea96, IKNP03], we require the sender and the receiver to execute k oblivious transfers in parallel. This is known as the batch setting and such protocols are called batch OT protocols.

A large body of work focused on minimizing the round-complexity of oblivious transfer (e.g., [NP01, AIR01, PVW08, DGH+20]). We now know constructions of round-optimal (i.e., two-round) oblivious transfer protocols in the CRS model that are UC-secure against malicious adversaries based on standard cryptographic assumptions such as CDH, LPN, QR, and LWE [PVW08, DGH+20].

Communication Complexity. Recently, the communication complexity of OT has received a lot of attention [IKNP03, BCGI18, BCG+19a, BCG+19b, DGI+19, BCG+20a, BCG+20b, GHO20, CGH+21, ADD+22]. This line of research has culminated with the works of Gentry et al. [GH19], Brakerski et al. [BDGM19, BBDP22], and Branco et al. [BDS23] which focused on minimizing the communication complexity of batch OT protocols. They gave constructions of two-round semi-honest and statistical sender private batch OT protocols with (near) optimal communication cost. Specifically, the communication complexity of these protocols was $2k + o(k) \cdot \mathsf{poly}(\lambda)$ where k is the batch size and λ is the security parameter. These works further argued that any batch OT protocol requires at least $2k$ bits to ensure correctness.

We observe that at least $3k$ bits of communication is needed to prove security against malicious adversaries via a black-box and straight-line simulator.[1] Indeed, the first-round message from the receiver should contain enough information for the simulator to extract all its choice bits and the sender's message should contain enough information for the simulator to extract both the sender's inputs. By standard incompressibility argument, it follows that the communication complexity should be at least $3k$ bits. Given this lower bound, a natural question is to give a construction with matching communication cost.

Prior Work. Though not explicitly stated, we observe that the techniques introduced in prior works can be used to construct a malicious-secure, two-round batch OT protocol with optimal communication.

1. **Spooky Encryption.** Spooky encryption [DHRW16] is a generalization of fully homomoprhic encryption with the following property. Let $(\mathsf{pk}_1, \mathsf{sk}_1)$ and $(\mathsf{pk}_2, \mathsf{sk}_2)$ be independently sampled public-key, secret-key pairs for a spooky encryption scheme. Given $\mathsf{Enc}(\mathsf{pk}_1, x_1)$, $\mathsf{Enc}(\mathsf{pk}_2, x_2)$ and a two-party function f, there is a special homomorphic operation that generates $\mathsf{Enc}(\mathsf{pk}_1, y_1)$ and $\mathsf{Enc}(\mathsf{pk}_2, y_2)$ such that $y_1 \oplus y_2 = f(x_1, x_2)$. Let us now see how to use spooky encryption to construct a batch OT protocol with optimal communication. In the first round, the receiver samples $(\mathsf{pk}_1, \mathsf{sk}_1)$ of a spooky encryption scheme and a PRG seed s_1. It generates $\mathsf{Enc}(\mathsf{pk}_1, s_1)$ and sends this ciphertext along with $\mathbf{b}' = \mathsf{PRG}(s_1) \oplus \mathbf{b}$ where $\mathbf{b}$ denotes the vector of choice bits of the receiver. In the second round, the sender samples $(\mathsf{pk}_2, \mathsf{sk}_2)$ of the spooky encryption scheme and PRG seed s_2. It generates $\mathsf{Enc}(\mathsf{pk}_2, s_2)$. Let f be a two-party function that takes in s_1 and s_2 and computes $\mathsf{PRG}(s_1) \circ \mathsf{PRG}(s_2)$ where $\circ$ denotes the point-wise product of the two strings. The sender homomorphically evaluates f on $\mathsf{Enc}(\mathsf{pk}_1, s_1)$ and $\mathsf{Enc}(\mathsf{pk}_2, \mathsf{sk}_2)$ to obtain $\mathsf{Enc}(\mathsf{pk}_1, \mathbf{y}_1)$ and $\mathsf{Enc}(\mathsf{pk}_2, \mathbf{y}_2)$ such that $\mathbf{y}_1 \oplus \mathbf{y}_2 = f(s_1, s_2)$. It decrypts the second ciphertext using sk_2 to learn $\mathbf{y}_2$. Note that if the sender sends $\mathsf{Enc}(\mathsf{pk}_2, s_2)$ to the receiver, the receiver can obtain $\mathbf{y}_1$. Observe that $\mathbf{y}_1 = \mathbf{y}_2 \oplus \mathsf{PRG}(s_1) \circ \mathsf{PRG}(s_2)$. Hence, we can use this to generate random OT correlations where the receiver's choice bits are given by $\mathsf{PRG}(s_1)$ and the sender's random input bits are $(\mathbf{y}_2, \mathbf{y}_2 \oplus \mathsf{PRG}(s_2))$. Further, the sender can derandomize this random OT correlation to actual OT using the receiver's first round message $\mathbf{b}'$. The receiver can then use $\mathbf{y}_1$ to recover the inputs of the sender corresponding to its choice bits $\mathbf{b}$. To protect against malicious adversaries, we add additional NIZK proofs (that are known from LWE [PS19]) showing that the receiver's and the sender's encryptions are generated correctly. This protocol has a total communication cost of $3k + \mathsf{poly}(\lambda)$ bits. Dodis et al. [DHRW16] gave a construction of spooky FHE under LWE with super-polynomial modulus-to-noise ratio.

2. **Pseudorandom Correlation Generators.** Orlandi et al. [OSY21] constructed a non-interactive protocol for generating the seeds of a pseudorandom correlation generator (PCG) for computing correlated oblivious transfer. These short seeds can be locally expanded to give many instances of correlated OT.[2] These can be further transformed to standard OT correlations using a correlation-robust hash function [IKNP03]. Generically, the entire protocol requires at least three rounds in the CRS model (one round to generate the seeds and two more rounds to derandomize from random OT correlations to actual OTs). However, if we open up the protocol of [OSY21], we note that the OT protocol can in fact be implemented in two rounds as the receiver can already compute its random choice bits after the CRS is known. One caveat, though, is that [OSY21] requires a way to publicly generate random Goldwasser-Micali ciphertexts encrypting random bits. This would either need a random oracle, or a uniform CRS with length proportional to the number of OTs which is non-reusable. Boyle et al. [BCG+19a] gave a two-round protocol for generating many instances of correlated OTs with small communication using LPN in the random oracle model.

To conclude, we know how to build optimal-rate maliciously-secure OT in two rounds in the standard model i) from LWE with super-polynomial modulus to noise ratio, ii) group based assumptions (namely DCR and

[1] Straight-line simulation is required to show UC-security.

[2] Correlated OTs are generated by sampling a fixed offset $\Delta \leftarrow \{0,1\}^\lambda$ and generating several instances of the form $(b, v + b \cdot \Delta)$ where $b \leftarrow \{0,1\}$ and $v \leftarrow \{0,1\}^\lambda$. The receiver gets $(b, v + b \cdot \Delta)$ and the sender gets (v, Δ).

QR) with a large CRS (that is, a CRS which grows with the batch size) that is non-reusable, or a random oracle, or (iii) LPN assumption in the random oracle model.

Given the above state-of-the-art, the main question we would like to address in this work is the following:

Can we construct a malicious-secure, two-round batch oblivious transfer protocol with (near) optimal communication cost with a short resuable CRS under assumptions that are weaker than LWE?

Our Results. We answer the above question affirmatively and give a construction of a two-round batch OT protocol in the CRS model with (near) optimal communication and satisfying UC-security against malicious adversaries. The security of our construction is based on the Quadratic Residuosity (QR) and Learning Parity with Noise (LPN) assumptions. Formally,

Theorem 1. *Assume the hardness of Quadratic Residuosity and the Learning Parity with Noise. There exists a two-round protocol for computing batch of k oblivious transfers in the CRS model with total communication cost of $3k + o(k) \cdot \mathsf{poly}(\lambda)$ bits when the sender's private inputs are bits and $k + 2km + o(k) \cdot \mathsf{poly}(\lambda)$ bits when the sender's private inputs are strings of length m. The protocol achieves UC-security against malicious adversaries.*

2 Technical Overview

A recent line of work [GH19, BDGM19, BBDP22, BDS23] studies the hardness assumptions needed to construct communication-optimal 2-message batch OT, that is batch OT protocols in which the amortized communication cost per OT approaches 2 bits. Such protocols cannot be maliciously secure under straight-line simulation, as the total amount of communication is insufficient to encode both the sender's and receiver's inputs in an information-theoretic sense.

2.1 Warmup: The PVW Protocol

The starting point of our construction is the well-known protocol of Peikert, Vaikuntanathan and Waters [PVW08], in the following referred to as the PVW protocol. We will consider a simple variant of the PVW protocol in the QR setting which can be described as follows on a high-level [3]. Let $N = pq$ be a product of two safe primes and let $g \in \mathbb{QR}_N$ be a random generator of the group of quadratic residues in $\mathbb{Z}_N^*$, and let $u \in \mathbb{J}_N$ be a uniformly random element with Jacobi symbol 1. The common reference string consists of the N, g and u.

The receiver, given a choice bit $b \in \{0,1\}$ picks a random $x \leftarrow\!\!\$\ \mathbb{Z}_N$, sets $h = g^x \cdot u^b$ and sends h to the sender. The sender, on input two messages $m_0, m_1 \in \{0,1\}$ picks a random $r \leftarrow\!\!\$\ \mathbb{Z}_N$ and computes $c = g^r$, $c_0 = h^r \cdot (-1)^{m_0}$ and $c_1 = (h \cdot u^{-1})^r \cdot (-1)^{m_1}$. He then sends c, c_0 and c_1 to the receiver, who can recover $(-1)^{m_b}$ (and thus m_b) by computing c_b / c^x.

Security against a malicious receiver can (roughly) be argued by choosing the value u in the CRS as $u = g^y \cdot (-1)$ (for a random $y \leftarrow\!\!\$\ \mathbb{Z}_N^*$), which is indistinguishable from a random $u \in \mathbb{Z}_N^*$ under the QR assumption. Given the factorization $N = pq$ as a trapdoor, a simulator can now extract the choice bit b from h by testing whether h is a quadratic residue. Finally, we can argue that the values $c = g^r$, c_0 and c_1 statistically hide m_{1-b} as

$$c_0 = h^r(-1)^{m_0} = (g^r)^{x+by} \cdot (-1)^{br+m_1}$$
$$c_1 = (h \cdot u^{-1})^r \cdot (-1)^{m_1} = (g^r)^{x+(1-b)y} \cdot (-1)^{(1-b)r+m_1}.$$

We can rewrite this as

$$c_b = (g^r)^x \cdot (-1)^{m_b}$$
$$c_{1-b} = (g^r)^{x+y} \cdot (-1)^{r+m_{1-b}}.$$

[3]The QR-based construction in [PVW08] is based on Cocks' cryptosystem, but we will describe a version here based on the Brakerski Goldwasser encryption scheme [BG10]

Since p and q are safe primes, it holds that the group order of $\mathbb{QR}_N$ is odd, from which is follows that $(-1)^r$ is uniformly random in $\{-1,1\}$ given g^r as $g \in \mathbb{QR}_N$. Hence, the distribution of c_{1-b} is independent of m_{1-b} and the claim follows.

Security against a malicious sender can be argued by modifying the CRS such that $u = g^y$ is a random quadratic residue, which is indistinguishable from a random u under the QR assumption. Hence, the term h now statistically hides b. The senders input can be extracted by decrypting c_0 and c_1 using c, x and y.

The communication complexity of this protocol is far from constant; for a single bit OT, the receiver needs to send 1 group element, whereas the sender transmits 3 group elements. Using ciphertext compression techniques introduced in [DGI$^+$19] (which we discuss in the next paragraph), the sender's communication complexity may further be improved to 1 single group element and two additional bits.

To improve the communication complexity further, we will need to consider the batch setting, i.e. a setting in which the protocol parties run many independent OT instances in parallel.

2.2 Batch OT with Trapdoor Hash Functions

To make progress towards optimal communication complexity for the sender, in other words optimal download complexity, we will consider a batch variant of the PVW protocol based on a primitive called *trapdoor hash functions (TDH)* [DGI$^+$19].

Trapdoor Hash Functions In broad terms, a trapdoor hash function is a hash function that supports the generation of additional *hinting keys* that encode a secret function f. Using this hinting key and the hash key, a hash evaluator can release fine-grained *hints* along with a hash value of some input x. Using the hash value, the hints, and secret state needed to generate the hinting key, one can recover the output of f applied on x. The important efficiency requirement is that the size of the hints only grow with the length of the output of f and are otherwise, independent of the input length.

In a bit more detail, TDH consists of a setup algorithm Setup, which produces a public hashing key hk, and a hashing algorithm H which takes hk and an input x and produces a succinct hash value h. In addition to these, there are three more algorithms KeyGen, Enc and Dec with the following syntax.

- KeyGen takes as input a hashing key hk and some function $f : \{0,1\}^n \to \{0,1\}$ *which outputs a single bit*, and produces an hinting key ek and a corresponding trapdoor td.
- The encryption or hinting algorithm Enc takes a hinting key ek and an input $\mathbf{x}$ and produces a *binary* hint $z \in \{0,1\}$.
- The decryption algorithm Dec takes a trapdoor td and a hash value h and produces an offset $v \in \{0,1\}$.

In terms of security, we only require that the hinting key ek computationally hides the function f. In terms of correctness, we require that it holds for all supported functions f and all inputs $\mathbf{x}$ that

$$\mathsf{Enc}(\mathsf{ek},\mathbf{x}) = \mathsf{Dec}(\mathsf{td},\mathsf{H}(\mathsf{hk},\mathbf{x})) \oplus f(\mathbf{x}),$$

where $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, n)$ and $(\mathsf{ek},\mathsf{td}) \leftarrow \mathsf{KeyGen}(\mathsf{hk}, f)$. In other words, $\mathsf{Enc}(\mathsf{ek},\mathbf{x})$ produces a simple XOR-encryption of $f(\mathbf{x})$ under a key which can be recovered from $\mathsf{H}(\mathsf{hk},\mathbf{x})$ using the trapdoor td. This property can also be interpreted as the ability to *perfectly simulate* the hint $\mathsf{Enc}(\mathsf{ek},\mathbf{x})$ given only the succinct hash value $\mathsf{H}(\mathsf{hk},\mathbf{x})$, the trapdoor td and the bit $f(\mathbf{x})$. Conversely, it allows us to perfectly simulate $\mathsf{Dec}(\mathsf{td},\mathbf{x})$ without td, given only $h = \mathsf{H}(\mathsf{hk},\mathbf{x})$, ek, $f(x)$ and $\mathbf{x}$.

Döttling et al. [DGI$^+$19] provide a number of instantiations of TDH from DDH, LWE and QR. Looking ahead, in this work our goal is to construct *maliciously secure* OT. For this reason we will require the above correctness property of TDH to hold perfectly, which is the case for the QR-based construction in [DGI$^+$19], but not for their DDH-based construction.

We will thus briefly discuss the QR-based TDH of [DGI$^+$19]. The setup algorithm Setup generates an RSA-modulus $N = p \cdot q$ with two random safe primes p and q. It further generates n random quadratic residues $g_1, \dots, g_n \in \mathbb{QR}_N$. The hashing key hk consists of N and $g_1, \dots, g_n$. To hash a message $\mathbf{x} \in \{0,1\}^n$, the hash algorithm H computes

$$h = \prod_{i=1}^{n} g_i^{x_i}.$$

This construction only supports $\mathbb{F}_2$-linear functions $f : \{0,1\}^n \to \{0,1\}$, i.e. inner product functions $f(x) = \langle \mathbf{t}, \mathbf{x} \rangle$ for some vector $\mathbf{t} \in \mathbb{F}_2^n$. The key generation algorithm KeyGen, given a hashing $\mathsf{hk} = (N, g_1, \dots, g_n)$ and such a vector $\mathbf{t} = (t_1, \dots, t_n)$ (describing the function $f(\mathbf{x}) = \langle \mathbf{t}, \mathbf{x} \rangle$) chooses a uniformly random $r \in \mathbb{Z}_n$ and computes $h_1 = g_1^r \cdot (-1)^{t_1}, \dots, h_n = g_n^r \cdot (-1)^{t_n}$. The hinting key ek is then given by $h_1, \dots, h_n$, whereas the trapdoor td is the value r. We can routinely argue that ek hides $\mathbf{t}$ via the QR assumption.

To construct the hinting algorithm Enc, we will make use of an efficient "rounding" or "shrinking" algorithm $\mathsf{Shrink} : \mathbb{Z}_N^* \to \{0,1\}$ with the property that for all $h \in \mathbb{Z}_N^*$ it holds that $\mathsf{Shrink}(h \cdot (-1)) = \mathsf{Shrink}(h) \oplus 1$. Now, given $\mathsf{ek} = (h_1, \dots, h_n)$ and $\mathbf{x}$ the hinting algorithm computes and outputs

$$c = \mathsf{Shrink}\left(\prod_{i=1}^{n} h_i^{x_i} \right).$$

The decryption algorithm, given a hash value h and a trapdoor $\mathsf{td} = r$ computes and outputs $v = \mathsf{Shrink}(h^r)$. To see that this construction satisfies the correctness property, note that

$$\begin{aligned}
\mathsf{Enc}(\mathsf{ek}, \mathbf{x}) &= \mathsf{Shrink}\left(\prod_{i=1}^{n} h_i^{x_i} \right) \\
&= \mathsf{Shrink}\left(\prod_{i=1}^{n} (g_i^r \cdot (-1)^{t_i})^{x_i} \right) \\
&= \mathsf{Shrink}\left(\left(\prod_{i=1}^{n} g_i^{x_i} \right)^r \cdot (-1)^{\sum_{i=1}^{n} t_i x_i} \right) \\
&= \mathsf{Shrink}\left(h^r \cdot (-1)^{\langle \mathbf{t}, \mathbf{x} \rangle} \right) \\
&= \mathsf{Shrink}\left(h^r \right) \oplus \langle \mathbf{t}, \mathbf{x} \rangle \\
&= \mathsf{Dec}(\mathsf{td}, h) \oplus \langle \mathbf{t}, \mathbf{x} \rangle.
\end{aligned}$$

PVW with Trapdoor Hash Functions We will now discuss a new batched variant of the PVW protocol using TDH which achieves optimal communication complexity for the sender. The new protocol proceeds as follows.

The CRS consist of a hashing key hk for a TDH that supports inner product functions (such as the one discussed in the last paragraph) and m uniformly random vectors $\mathbf{t}_1, \dots, \mathbf{t}_m \in \mathbb{F}_2^n$.

The receiver, on input choice bits $b_1, \dots, b_m$ generates hinting keys and trapdoors $(\mathsf{ek}_i, \mathsf{td}_i) \leftarrow \mathsf{KeyGen}(\mathsf{hk}, b_i \cdot \mathbf{t}_i)$ for $i = 1, \dots, m$ and sends $\mathsf{ek}_1, \dots, \mathsf{ek}_m$ to the sender.

The sender, on input pairs of bits $(m_{1,0}, m_{1,1}), \dots, (m_{m,0}, m_{m,1})$ and given the hashing key hk and the hinting keys $\mathsf{ek}_1, \dots, \mathsf{ek}_m$ proceeds as follows. He picks a uniformly random $\mathbf{r} \in \mathbb{F}_2^n$ and computes $h \leftarrow \mathsf{H}(\mathsf{hk}, \mathbf{r})$. He then proceeds to compute

$$\begin{aligned}
w_{i,0} &\leftarrow \mathsf{Enc}(\mathsf{ek}_i, \mathbf{r}) \oplus m_{i,0} \\
w_{i,1} &\leftarrow \mathsf{Enc}(\mathsf{ek}_i, \mathbf{r}) \oplus \langle \mathbf{t}_i, \mathbf{r} \rangle \oplus m_{i,1},
\end{aligned}$$

for $i = 1, \dots, m$. He then sends the hash value h and the pairs $(w_{1,0}, w_{1,1}), \dots, (w_{m,0}, w_{m,1})$ to the receiver.

The receiver then computes and outputs $m_i' \leftarrow \mathsf{Dec}(\mathsf{td}_i, h) \oplus w_{i,b_i}$ for $i = 1, \dots, m$. We will first argue correctness of this protocol. Note that it holds that

$$\begin{aligned} m'_i &= \mathsf{Dec}(\mathsf{td}_i, h) \oplus w_{i,b_i} \\ &= \mathsf{Dec}(\mathsf{td}_i, h) \oplus \mathsf{Enc}(\mathsf{ek}_i, \mathbf{r}) \oplus b_i \cdot \langle \mathbf{t}_i, \mathbf{r} \rangle \oplus m_{i,b_i} \\ &= m_{i,b_i}, \end{aligned}$$

as $\mathsf{Enc}(\mathsf{ek}_i, \mathbf{r}) = \mathsf{Dec}(\mathsf{td}_i, h) \oplus \langle b_i \cdot \mathbf{t}_i, \mathbf{r} \rangle$ by the correctness of the TDH.

This protocol is not readily maliciously secure. However, assume for a moment there was an additional mechanism in place against a malicious sender which ensures that $h = \mathsf{H}(\mathsf{hk}, \mathbf{r})$ for some $\mathbf{r}$ without revealing $\mathbf{r}$, and further enables a simulator to extract $\mathbf{r}$. This could in principle be achieved via (designated verifier) zero-knowledge proofs of knowledge. Given $\mathbf{r}$, we can simulate the receiver's behavior (without knowledge of the b_i or td_i) by extracting the messages

$$\begin{aligned} m'_{i,0} &\leftarrow \mathsf{Enc}(\mathsf{ek}_i, \mathbf{r}) \oplus w_{i,0} \\ m'_{i,1} &\leftarrow \mathsf{Enc}(\mathsf{ek}_i, \mathbf{r}) \oplus \langle \mathbf{t}_i, \mathbf{r} \rangle \oplus w_{i,1} \end{aligned}$$

for all $i = 1, \ldots, m$. By the correctness property of the TDH it holds that

$$\mathsf{Dec}(\mathsf{td}_i, h) \oplus w_{i,b_i} = \mathsf{Enc}(\mathsf{ek}_i, \mathbf{r}) \oplus b_i \langle \mathbf{t}_i, \mathbf{r} \rangle \oplus w_{i,b_i} = m'_{i,b_i},$$

i.e. we can replace the receiver's output by m'_{i,b_i}, and this modification is perfectly indistinguishable from the receivers view. At this point, the simulation does not rely on the b_i or td_i anymore. Hence, we can use the function hiding property of the TDH to generate the hinting keys ek_i independently of the b_i, and hence simulate the protocol.

In turn, to establish security against a malicious receiver we need an additional mechanism which ensures well-formedness of hk and the ek_i and lets us extract the b_i and td_i from the receiver. Once such a mechanism is in place, we can argue security against a malicious receiver as follows. Note that by the correctness of the TDH a simulator in possession of the trapdoors td_i can equivalently compute the ciphertexts $w_{i,b}$ (for $i = 1, \ldots, m$ and $b \in \{0, 1\}$) via $w_{i,b} = \mathsf{Dec}(\mathsf{td}_i, h) \oplus (b \oplus b_i) \cdot \langle \mathbf{t}_i, \mathbf{r} \rangle \oplus m_{i,b}$. Consequently, for the message $m_{i,1-b_i}$, which should remain hidden from the receiver, it holds that

$$w_{i,1-b_i} = \mathsf{Dec}(\mathsf{td}_i, h) \oplus \langle \mathbf{t}_i, \mathbf{r} \rangle \oplus m_{i,1-b_i}.$$

That is, from the receiver's view the message $m_{i,1-b_i}$ is masked by the term $\langle \mathbf{t}_i, \mathbf{r} \rangle$. Note that the succinct hash value h is the only additional information the receiver learns about $\mathbf{r}$ besides the inner products $\langle \mathbf{t}_i, \mathbf{r} \rangle$. Furthermore, note that the hash value $h = \mathsf{H}(\mathsf{hk}, \mathbf{r})$ does not depend on the random vectors $\mathbf{t}_1, \ldots, \mathbf{t}_m$. Hence, given that m is sufficiently smaller than $n - \mathsf{bitlength}(h)$, we can appeal to the leftover hash lemma [DRS04, Reg05] to argue that the inner products $\langle \mathbf{t}_i, \mathbf{r} \rangle$ are statistically close to uniform given h, and consequently we can simulate the $w_{i,1-b_i}$ by choosing them uniformly at random.

While this protocol makes progress towards our goal of optimal *maliciously secure* OT, it has several glaring problems.

Most obviously, we have not specified how the additional extraction mechanism require for simulation affect the communication complexity of the protocol. We will postpone the discussion of this issue as there is another, far more severe issue with the above protocol: With the parameter choice discussed above we can achieve optimal amortized download complexity for the sender, however we have done so at the expense of the upload complexity! Specifically, in order to be able to extract a sufficient number of masks, we need to make n sufficiently larger than m. But looking at the TDH construction from QR above, this means that each hinting key ek_i consists of n group elements, while at the same time the receiver sends m hinting keys, which means in order to get m bit OTs, the receiver needs to send $\Omega(m^2)$ group elements. This means the total communication complexity of this protocol is *asymptotically worse* than repeating the simple QR-based PVW protocol above m times!

The underlying issue here is that our information theoretic argument for sender security runs into an entropy barrier; we cannot extract more than n random bits from $\mathbf{r}$. Consequently, to bypass this barrier and make this approach work we need to settle on a computational argument, that is we need to derive computationally secure masks from $\mathbf{r}$.

2.3 Computational Sender Security via LPN

This is where the *Learning Parity with Noise* (LPN) assumption comes into play. The basic observation is that, using nearly the same construction as above, the LPN assumption lets us extract more entropy than there is if we are able to deal with additional errors. The *decisional LPN assumption* postulates that for any polynomial bound m the samples $(\mathbf{t}_1, \langle \mathbf{t}_1, \mathbf{r} \rangle + f_1), \ldots, (\mathbf{t}_m, \langle \mathbf{t}_m, \mathbf{r} \rangle + f_m)$ are pseudorandom, where the $\mathbf{r} \leftarrow\$ \mathbb{F}_2^n$ and the $\mathbf{t}_i \leftarrow\$ \mathbb{F}_2^n$ are chosen uniformly random and the f_i are independent and follow a Bernoulli distribution, that is each f_i is 0 with probability $1 - \delta$ and 1 with probability δ. We will choose a slightly sub-constant δ, hence there will not be too many faulty positions.

A moment of reflection exposes that we cannot simply add noise terms f_i into the construction in the last paragraph, as LPN requires the secret $\mathbf{r}$ to be uniform, whereas $\mathbf{r}$ in the construction above "loses" entropy as the hash value $h = \mathsf{H}(\mathsf{hk}, \mathbf{r})$ is leaked. However, we have also seen in the last paragraph that we can use the leftover hash lemma to extract a uniformly random vector from $\mathbf{r}$ in the presence of the leakage $h = \mathsf{H}(\mathsf{hk}, \mathbf{r})$. Hence, we will modify the construction such that it first extracts an LPN secret $\hat{\mathbf{r}}$ from $\mathbf{r}$, and then expands $\hat{\mathbf{r}}$ via LPN. We can achieve this via a single linear function!

We will now discuss the necessary modifications to the protocol in the last paragraph in more detail. Let n' be sufficiently smaller than $n - \mathsf{bitlength}(h)$. We will first discuss how CRS generation has to be modified. Instead of choosing the $\mathbf{t}_i \in \mathbb{F}_2^n$ uniformly random, we will choose a uniformly random matrix $\mathbf{V} \in \mathbb{F}_2^{n' \times n}$ and uniformly random vectors $\hat{\mathbf{t}}_1, \ldots, \hat{\mathbf{t}}_m$ and set $\mathbf{t}_i \leftarrow \mathbf{V}^\top \cdot \hat{\mathbf{t}}_i$ for $i = 1, \ldots, m$. Note that this is equivalent to putting the random matrix $\mathbf{V}$ and the random vectors $\hat{\mathbf{t}}_1, \ldots, \hat{\mathbf{t}}_m$ into the CRS and letting the parties compute the $\mathbf{t}_i$.

Both the receiver's first and second phase are as before, we will only modify the sender's algorithm. As before, the sender picks a uniformly random $\mathbf{r} \in \mathbb{F}_2^n$ and computes $h \leftarrow \mathsf{H}(\mathsf{hk}, \mathbf{r})$ and $z_i \leftarrow \mathsf{Enc}(\mathsf{ek}_i, \mathbf{r})$ for $i = 1, \ldots, m$. However, the computation of the $w_{i,b}$ is now modified by introducing noise terms. Specifically, for $i = 1, \ldots, m$ the sender computes

$$\begin{aligned} w_{i,0} &\leftarrow z_i \oplus f_{i,0} \oplus m_{i,0} \\ w_{i,1} &\leftarrow z_i \oplus \langle \mathbf{t}_i, \mathbf{r} \rangle \oplus f_{i,1} \oplus m_{i,1}, \end{aligned}$$

where the $f_{i,b}$ are chosen from a Bernoulli distribution with parameter δ. As before, he then sends the hash value h and the pairs $(w_{1,0}, w_{1,1}), \ldots, (w_{m,0}, w_{m,1})$ to the receiver.

We will first analyze correctness of this protocol. Letting $m_i' \leftarrow \mathsf{Dec}(\mathsf{td}_i, h) \oplus w_{i,b_i}$ for $i = 1, \ldots, m$ be the receiver's outputs, we can establish using the correctness of the TDH that

$$m_i' = m_{i,b_i} \oplus f_{i,b_i},$$

which means that each output of the receiver will be faulty with (small) probability δ. We will later describe a mechanism that deals with the errors and merely observe now that if the total number of errors is sufficiently small, we will be able to afford a relatively costly mechanism (in terms of communication complexity) to correct the errors.

At this point we will only examine security against malicious receivers. We do this in a few hybrid steps. As before, via the correctness property of the TDH we can simulate the $w_{i,b}$ via

$$\begin{aligned} w_{i,b_i} &\leftarrow \mathsf{Dec}(\mathsf{td}_i, h) \oplus f_{i,b_i} \oplus m_{i,b_i} \\ w_{i,1-b_i} &\leftarrow \mathsf{Dec}(\mathsf{td}_i, h) \oplus \langle \mathbf{t}_i, \mathbf{r} \rangle \oplus f_{i,1-b_i} \oplus m_{i,1-b_i}. \end{aligned}$$

First note that it holds by our choice of the $\mathbf{t}_i = \mathbf{V}^\top \hat{\mathbf{t}}_i$ that

$$\langle \mathbf{t}_i, \mathbf{r} \rangle = \langle \mathbf{V}^\top \hat{\mathbf{t}}_i, \mathbf{r} \rangle = \langle \hat{\mathbf{t}}_i, \mathbf{V}\mathbf{r} \rangle.$$

That is, we can equivalently compute $w_{i,1-b_i}$ via

$$w_{i,1-b_i} \leftarrow \mathsf{Dec}(\mathsf{td}_i, h) \oplus \langle \hat{\mathbf{t}}_i, \mathbf{V}\mathbf{r} \rangle \oplus f_{i,1-b_i} \oplus m_{i,1-b_i}.$$

As before, noting that $\mathbf{r}$ has high conditional min-entropy given the succinct hash $h = \mathsf{H}(\mathsf{hk}, \mathbf{r})$ and that h is independent of $\mathbf{V}$, we can invoke the leftover hash lemma and replace $\mathbf{Vr}$ with a uniformly random $\hat{\mathbf{r}} \leftarrow\!\!\$\ \mathbb{F}_2^{n'}$ while introducing only a negligible statistical distance in the view of a malicious receiver. That is, we compute $w_{i,1-b_i}$ via

$$w_{i,1-b_i} \leftarrow \mathsf{Dec}(\mathsf{td}_i, h) \oplus \langle \hat{\mathbf{t}}_i, \hat{\mathbf{r}} \rangle \oplus f_{i,1-b_i} \oplus m_{i,1-b_i}.$$

Now, as $\hat{\mathbf{r}}$ is independent of h, we can rely on the pseudorandomness of LPN to replace the $\langle \hat{\mathbf{t}}_i, \hat{\mathbf{r}} \rangle \oplus f_{i,1-b_i}$ with uniformly random and independent $u_i \leftarrow\!\!\$\ \{0,1\}$. That is, we compute $w_{i,1-b_i}$ via

$$w_{i,1-b_i} \leftarrow \mathsf{Dec}(\mathsf{td}_i, h) \oplus u_i \oplus m_{i,1-b_i}.$$

But this means that $w_{i,1-b_i}$ itself is independently uniformly random in $\{0,1\}$, i.e. we can equivalently just choose $w_{i,1-b_i} \leftarrow\!\!\$\ \{0,1\}$. We can conclude that in this final hybrid the sender's message is independent of the $m_{i,1-b_i}$, and we have thus established security against malicious receivers (given that we have a mechanism to extract the td_i and b_i).

We will now briefly consider the communication complexity of our current protocol. The modification discussed in this paragraph (omitting the issue of errors) does not affect the download communication complexity, the sender still sends (amortized) 2 bits per OT. However, the upload communication complexity is now just $O(n \cdot m)$ group elements, where $n = \mathsf{poly}(\lambda)$ is a fixed polynomial, independent of m.

2.4 Key-Homomorphic Trapdoor Hash Functions

We will now discuss an additional mechanism which lets us further compress the upload communication complexity. The starting observation here is that each hinting key ek_i only "encrypts" a single bit b_i, but consists of n group elements. To address this issue, we will pursue a *hybrid encryption approach* similar to [BBDP22], i.e. we will encrypt the bits b_i under a symmetric key encryption scheme with optimal rate along with TDH hinting keys which encrypt the corresponding secret key, and then let the sender *homomorphically expand* these ciphertexts into the actual hinting keys ek_i.

Consequently, to implement this approach we need to add homomorphic capabilities to our underlying trapdoor hash functions for linear functions. We call this primitive *key-homomorphic trapdoor hash functions.* Recall that our trapdoor hash functions allow for hints which let the receiver learn linear functions $f(\mathbf{r}) = \langle \mathbf{t}, \mathbf{r} \rangle$ of then sender's input $\mathbf{r}$, i.e. it holds that

$$\mathsf{Enc}(\mathsf{ek}, \mathbf{r}) = \mathbf{Dec}(\mathsf{td}, \mathsf{H}(\mathsf{hk}, \mathbf{r})) \oplus \langle \mathbf{t}, \mathbf{r} \rangle,$$

where $(\mathsf{ek}, \mathsf{td}) \leftarrow \mathsf{KeyGen}(\mathsf{hk}, \mathbf{t})$.

In a key-homomorphic trapdoor hash function, we require keys to support homomorphic operations, that is, given ek_1 and ek_2 with $(\mathsf{ek}_1, \mathsf{td}_1) \leftarrow \mathsf{KeyGen}(\mathsf{hk}, \mathbf{t}_1)$ and $(\mathsf{ek}_2, \mathsf{td}_2) \leftarrow \mathsf{KeyGen}(\mathsf{hk}, \mathbf{t}_2)$, one can efficiently derive a key ek^* corresponding to $\mathbf{t}_1 \oplus \mathbf{t}_2$, given only hk, ek_1 and ek_2, and likewise a trapdoor td^* from hk, td_1 and td_2 such that

$$\mathsf{Enc}(\mathsf{ek}^*, \mathbf{r}) = \mathbf{Dec}(\mathsf{td}^*, \mathsf{H}(\mathsf{hk}, \mathbf{r})) \oplus \langle \mathbf{t}_1 \oplus \mathbf{t}_2, \mathbf{r} \rangle.$$

More formally, we require a key-homomorphic evaluation algorithm Eval along with a corresponding decryption algorithm Dec' such that the following holds. Eval takes as input a vector $\mathbf{d} = (d_1, \ldots, d_k) \in \mathbb{F}_2^k$ and hinting keys $\mathsf{ek}_1, \ldots, \mathbf{ek}_k$ and outputs a hinting key ek^*, while Dec' also takes the vector $\mathbf{d}$ and trapdoors $\mathsf{td}_1, \ldots, \mathsf{td}_k$ as well as a hash value h and outputs a trapdoor td^*, such that it holds that

$$\mathsf{Enc}(\mathsf{Eval}(\mathbf{d}, \mathsf{ek}_1, \ldots, \mathsf{ek}_k), \mathbf{r}) = \mathsf{Dec}'(\mathbf{d}, \mathsf{td}_1, \ldots, \mathsf{td}_k, h) \oplus \left\langle \sum_{i=1}^{k} d_i \mathbf{t}_i, \mathbf{r} \right\rangle.$$

Turning to the construction of key-homomorphic TDH, we observe that the construction of TDH from QR [DGI+19] we discussed above readily supports key homomorphism. Specifically, given two hinting keys $\mathsf{ek}_1 = (h_{1,i} = g_i^{r_1} \cdot (-1)^{t_{1,i}})_{i \in [n]}$ and $\mathsf{ek}_2 = (h_{2,i} = g_i^{r_2} \cdot (-1)^{t_{2,i}})_{i \in [n]}$, it holds that

$$h_{1,i} \cdot h_{2,i} = g_i^{r_1} (-1)^{t_{1,i}} \cdot g_i^{r_2} (-1)^{t_{2,i}} = g_i^{r_1 + r_2} \cdot (-1)^{t_{1,i} + t_{2,i}},$$

i.e. $\mathsf{ek}^* = (h_{1,i} \cdot h_{2,i})_{i \in [n]}$ is a hinting key for $\mathbf{t}_1 \oplus \mathbf{t}_2$, and the corresponding trapdoor is $\mathsf{td}^* = r_1 + r_2$.

2.5 Compressing the Receiver's Message via LPN and Key-Homomorphic TDH

We will use the LPN assumption once more, this time in conjunction with the key-homomorphism property of the trapdoor hash function to implement a hybrid encryption approach which enables the receiver to transmit the encoding keys corresponding to his choice bits in a *compressed* form, which can then be locally expanded by the sender. In essence, this approach is an adaptation of the LPN-based compression mechanisms of [BBDP22]. Specifically, we can think of the hinting keys of a key-homomorphic TDH as ciphertexts of a homomorphic encryption scheme encrypting a vector $\mathbf{t}$. Following the blueprint of [BBDP22], we let the receiver provide TDH hinting keys which encode an LPN secret, and additionally provide LPN ciphertexts of the actual choice bits along with. Given this information, the sender will then be able to homomorphically derive hinting keys that encode the actual choice bits.

For simplicity, we start be considering a single $\mathbf{t}$. Assume the common reference string contains uniformly random vectors $\mathbf{d}_1, \dots, \mathbf{d}_\ell \in \mathbb{F}_2^n$. The receiver will choose an LPN secret $\mathbf{s} \leftarrow\!\!\$\, \mathbb{F}_2^n$ and compute hinting keys $\mathsf{ek}_0, \mathsf{ek}_1, \dots, \mathsf{ek}_n$, where ek_0 encodes $\mathbf{t}$ and ek_i encodes $s_i \cdot \mathbf{t}$ for $0 < i \leq n$. The receiver further encrypts his choice bits $b_1, \dots, b_\ell$ to

$$c_i \leftarrow \langle \mathbf{s}, \mathbf{d_i} \rangle + e_i + b_i$$

for $i = 1, \dots, k$. Here the $e_i \in \{0,1\}$ are chosen from a Bernoulli distribution with slightly sub-constant parameter ϵ, that is each e_i is independently 1 with probability ϵ and otherwise 0.

The receiver now sends sends the hinting keys $\mathsf{ek}_0, \mathsf{ek}_1, \dots, \mathsf{ek}_n$ and the ciphertexts $c_1, \dots, c_\ell$ to the sender. Sender can compute expanded encoding keys via

$$\mathsf{ek}'_j = \mathsf{Eval}((c_j, \mathbf{d}_j), \mathsf{ek}_0, \mathsf{ek}_1, \dots, \mathsf{ek}_n)$$

for $j = 1, \dots, \ell$. By the homomorphic correctness of the TDH, ek'_j is a hinting key for the vector

$$\begin{aligned} c_j \cdot \mathbf{t} + \sum_{i=1}^{n} d_{j,i} s_i \mathbf{t} &= (c_j + \langle \mathbf{s}, \mathbf{d} \rangle) \cdot \mathbf{t} \\ &= (\langle \mathbf{s}, \mathbf{d_j} \rangle + e_j + b_j + \mathbf{s}, \mathbf{d} \rangle) \cdot \mathbf{t} \\ &= (b_j + e_j) \cdot \mathbf{t}. \end{aligned}$$

Consequently, if $e_j = 0$, which holds except with sub-constant probability ϵ, the sender obtains the correct hinting key for $b_j \cdot \mathbf{t}$. The sender now proceeds analogous to the previous protocol. He chooses uniform vectors $\mathbf{r}_1, \dots, \mathbf{r}_\ell$ and computes hash values $h_j \leftarrow \mathsf{H}(\mathsf{hk}, \mathbf{r}_j)$ for $j = 1, \dots, \ell$. Furthermore, he computes

$$\begin{aligned} w_{j,0} &\leftarrow \mathsf{Enc}(\mathsf{ek}'_j, \mathbf{r}_j) \oplus f_{j,0} \oplus m_{j,0} \\ w_{j,1} &\leftarrow \mathsf{Enc}(\mathsf{ek}'_j, \mathbf{r}_j) \oplus \langle \mathbf{t}, \mathbf{r}_j \rangle \oplus f_{j,1} \oplus m_{j,1}, \end{aligned}$$

for $j = 1, \dots, \ell$. He then sends the $h_1, \dots, h_\ell$ and $(w_{j,0}, w_{j,1})_{j \in [\ell]}$ to the receiver.

Using the trapdoors $\mathsf{td}_0, \mathsf{td}_1, \dots, \mathsf{td}_n$ corresponding to $\mathsf{ek}_0, \mathsf{ek}_1, \dots, \mathsf{ek}_n$ the receiver then computes

$$m'_{j,b_j} \leftarrow \mathsf{Dec}'((c_j, \mathbf{d}_j), \mathsf{td}_0, \dots, \mathsf{td}_n, h_j) \oplus w_{j,b_j}$$

for $j = 1, \dots, \ell$. This concludes the outline of the protocol.

We will first look at correctness of this protocol. By the homomorphic correctness of TDH it holds that

$$\begin{aligned} m'_{j,b_j} =& \mathsf{Dec}'((c_j, \mathbf{d}_j), \mathsf{td}_0, \dots, \mathsf{td}_n, h_j) \oplus w_{j,b_j} \\ =& \mathsf{Dec}'((c_j, \mathbf{d}_j), \mathsf{td}_0, \dots, \mathsf{td}_n, \mathsf{H}(\mathsf{hk}, \mathbf{r}_j)) \\ & \oplus \mathsf{Enc}(\mathsf{Eval}((c_j, \mathbf{d}_j), \mathsf{ek}_0, \mathsf{ek}_1, \dots, \mathsf{ek}_n), \mathbf{r}_j) \oplus b_j \cdot \langle \mathbf{t}, \mathbf{r}_j \rangle \oplus f_{j,b_j} \oplus m_{j,b_j} \\ &= (b_j \oplus e_j) \cdot \langle \mathbf{t}, \mathbf{r}_j \rangle \oplus b_j \cdot \langle \mathbf{t}, \mathbf{r}_j \rangle \oplus f_{j,b_j} \oplus m_{j,b_j} \\ &= e_j \cdot \langle \mathbf{t}, \mathbf{r}_j \rangle \oplus f_{j,b_j} \oplus m_{j,b_j}. \end{aligned}$$

Consequently, if both e_j and f_{j,b_j} are 0, which happens except with (small) probability $\leq \delta + \epsilon$, the receiver obtains the correct output m_{j,b_j}.

For simplicity, we only considered a single vector $\mathbf{t}$ in this sketch; this is insufficient to get optimal download communication as the sizes of the hash values h_j are not amortized. Hence, in the full protocol we need to use all the vectors $\mathbf{t}_1, \dots, \mathbf{t}_m$.

The Full Protocol For the full protocol, it will be convenient to arrange both the receiver's choice bits and the sender's messages in a matrix form. That is, the receiver's choice bits are $b_{i,j}$ and the sender's messages are $(m_{i,j,0}, m_{i,j,1})$ for $i \in [m]$ and $j \in [\ell]$. I.e., we will get $m \cdot \ell$ batch OTs.

The full protocol is summarized as follows.

- The common reference string contains the hashing key hk, random vectors $\mathbf{t}_1, \dots, \mathbf{t}_m \in \mathbb{F}_2^n$ as in Section 2.3, and uniformly random $\mathbf{d}_{i,j} \in \mathbb{F}_2^n$ for $i \in [m]$ and $j \in [\ell]$.
- The receiver generates hinting keys $\mathsf{ek}_{i,k}$ encoding $s_k \cdot \mathbf{t}_i$ for $i \in [m]$ and $k \in [n]$ together with a corresponding trapdoor $\mathsf{td}_{i,k}$. The receiver further encrypts each $b_{i,j}$ to
$$c_{i,j} \leftarrow \langle \mathbf{s}, \mathbf{d}_{i,j} \rangle + e_{i,j} + b_{i,j}.$$
The receiver sends the $\mathsf{ek}_{i,k}$ and the $c_{i,j}$ to the sender.
- The sender homomorphically derives hinting keys $\mathsf{ek}'_{i,j}$ corresponding to $(b_{i,j} \oplus e_{i,j}) \cdot \mathbf{t}_i$ from the $\mathsf{ek}_{i,k}$, $\mathbf{d}_{i,j}$ and the $c_{i,j}$ as described above. The sender now chooses $\mathbf{r}_1, \dots, \mathbf{r}_\ell \leftarrow\!\!\$ \{0,1\}^n$ uniformly at random and computes $h_j \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathbf{r}_j)$. He further computes
$$w_{i,j,0} \leftarrow \mathsf{Enc}(\mathsf{ek}'_{i,j}, \mathbf{r}_j) \oplus f_{i,j,0} \oplus m_{i,j,0}$$
$$w_{i,j,1} \leftarrow \mathsf{Enc}(\mathsf{ek}'_{i,j}, \mathbf{r}_j) \oplus \langle \mathbf{t}_i, \mathbf{r}_j \rangle \oplus f_{i,j,1} \oplus m_{i,j,1},$$
for $i \in [m]$ and $j \in [\ell]$.
- The receiver recovers the $m'_{i,j,b_{i,j}}$ via
$$m'_{i,j,b_{i,j}} \leftarrow \mathsf{Dec}'(\mathsf{Eval}((c_{i,j}, \mathbf{d}_{i,j}), \mathsf{td}_{i,0}, \dots, \mathsf{td}_{i,n}), h_j) \oplus w_{i,j,b_{i,j}}.$$

The correctness analysis is identical to the one provided in the last paragraph, i.e. it holds that $m'_{i,j,b_{i,j}} = m_{i,j,b_{i,j}}$, except with probability $\delta + \epsilon$ over the choice of the e_j and $f_{i,j}$.

In terms of security, note the following. Once we extract the $\mathbf{r}_j$ and the locations of the errors $f_{i,j}$, we can simulate the output of the receiver without knowledge of the LPN secret $\mathbf{s}$ or the choice-bits $b_{i,j}$. Consequently, we can use the LPN assumption to replace the $c_{i,j}$ with uniformly random values.

In terms of security against a malicious receiver, we now have to extract the trapdoors $\mathsf{td}_{i,k}$ and the LPN secret $\mathbf{s}$, as well as the locations of the errors $e_{i,j}$. Once these are known, we can simulate the sender's messages without knowledge of the secrets $\mathbf{r}_1, \dots, \mathbf{r}_\ell$, but only the corresponding LPN samples $\langle \mathbf{t}_i, \mathbf{r}_j \rangle \oplus f_{i,j,1-b_{i,j}}$ as well as the hash-values $h_1, \dots, h_\ell$. The rest of the argument proceeds analogous to the argument given in Section 2.3.

Communication Complexity We will now take at look at the communication complexity of this protocol.

- The receiver's message consists of $m \cdot n$ hinting keys $\mathsf{ek}_{i,j}$ (each consisting of $O(n)$ group elements) and $m \cdot \ell$ bits $c_{i,j}$. Hence the upload rate of this protocol is
$$\rho_{up} = \frac{m \cdot n^2 \mathsf{poly}(\lambda) + m \cdot \ell}{m \cdot \ell} = 1 + \frac{n^2 \mathsf{poly}(\lambda)}{\ell} = 1 + \frac{\mathsf{poly}(\lambda)}{\ell},$$
as we can choose the LPN dimension n as a fixed polynomial in the security parameter.

- The sender's message consists of ℓ hash values h_j (each consisting of $O(1)$ group elements) and $2 \cdot m \cdot \ell$ bits $w_{i,j,0}$ and $w_{i,j,1}$. Hence the download rate becomes

$$\rho_{down} = \frac{\ell \cdot \mathsf{poly}(\lambda) + 2m \cdot \ell}{m \cdot \ell} = 2 + \frac{\mathsf{poly}(\lambda)}{m}.$$

Consequently, by choosing $\ell, m = \mathsf{poly}(\lambda)$ as sufficiently large polynomials, we will obtain asymptotically optimal rate.

2.6 Correcting Errors and achieving Malicious Security

We will now briefly discuss which additional mechanisms need to be deployed in order to make this protocol both *correct* and *maliciously secure*. Taking the analysis of the communication complexity in the last Section as a guideline, we can afford to "spend" additional $m \cdot \mathsf{poly}(\lambda)$ bits of communication in the receiver's message, and $\ell \cdot \mathsf{poly}(\lambda)$ bits on top of the sender's message.

The additional mechanism will ensure the following properties:

1. For every error location (i, j) for which $e_{i,j} = 1$ we need to "flip" the receiver's output.
2. For every error location $(i, j, b_{i,j})$ (where $b_{i,j}$ is the receiver's choice bit at i, j) where $f_{i,j,b_{i,j}} = 1$, we need to signal to the receiver that an error occurred at this location so that the corresponding output can be corrected.
3. For every "column" $j \in [\ell]$ we need to make the $\mathbf{r}_j$ corresponding to the $h_j = \mathsf{H}(\mathsf{hk}, \mathbf{r}_j)$ extractable. Furthermore, we need to make the LPN secret $\mathbf{s}$ and the *support* of the error vector $\mathbf{e}_j = (e_{1,j}, \dots, e_{m,j})$ extractable. Since $\mathbf{e}_i$ has Hamming weight at most $2\epsilon m$, we can describe it via a succinct list of indices of length $2\epsilon m$. Furthermore, we need to make the the support of the error vectors $\mathbf{f}_{j,0} = (f_{1,j,0}, \dots, f_{m,j,0})$ and $\mathbf{f}_{j,1} = (f_{1,j,1}, \dots, f_{m,j,1})$ extractable. Since each of this vectors has Hamming weight at most $2\delta m$, we can describe each of them via a list of length $2\delta m$.
4. For every "row" $i \in [m]$ and for $k \in \{0, \dots, n\}$ we need to make the trapdoors $\mathsf{td}_{i,k}$ corresponding to the hinting keys $\mathsf{ek}_{i,k}$ extractable and ensure that the $\mathsf{ek}_{i,k}$ were generated correctly with respect to the LPNs secret $\mathbf{s}$, and the vectors $\mathbf{t}_i$ which are taken from the common reference string.

Property 4 can be achieved succinctly via a computationally secure rate-1 conditional disclosure of secrets (CDS) scheme, which can routinely implemented from general purpose non-interactive secure computation (NISC) protocol (with malicious UC-security) and pseudorandom functions. This protocol can be run "piggy-back" style with the main protocol, so it does not increase the round complexity.

We will thus focus on properties 1,2 and 3. We will address these issues *simultaneously* using also a general purpose NISC protocol with malicious UC-security. The key insight is that both our corrections and checks do not need to be performed "globally", but merely "column"-wise. Hence the NISC for each column can be succinct. Similarly important, these 2-message NISC protocols can also be piggy-backed with the main protocol, so they will neither increase the round complexity.

We will need an additional mechanism which lets us correct the LPN errors $e_{i,j}$ on the receiver's choice-bits. A mechanism achieving this was recently proposed in [BBDP22]. For every column j the sender chooses a fresh key K_j for a puncturable pseudorandom function [BW13, KPTZ13, BGI14] which is puncturable in $2\epsilon m$ positions. We need to modify the protocol such that the $w_{i,j,0}$ and $w_{i,j,1}$ are computed via

$$w_{i,j,0} \leftarrow \mathsf{Enc}(\mathsf{ek}'_{i,j}, \mathbf{r}_j) \oplus f_{i,j,0} \oplus m_{i,j,0} \oplus \mathsf{PRF}_{K_j}(i)$$
$$w_{i,j,1} \leftarrow \mathsf{Enc}(\mathsf{ek}'_{i,j}, \mathbf{r}_j) \oplus \langle \mathbf{t}_i, \mathbf{r}_j \rangle \oplus f_{i,j,1} \oplus m_{i,j,1} \oplus \mathsf{PRF}_{K_j}(i).$$

Furthermore, we will need to make the vectors $\mathbf{t}_1, \dots, \mathbf{t}_m$, which are part of the CRS, available to the NISC computation. However, since we need to NISC protocol to have sublinear communication complexity in m, we cannot make the $\mathbf{t}_i$ explicit inputs to the NISC. However, each NISC_j will only need to access a few of the $\mathbf{t}_i$. Hence, we will bind to the $\mathbf{t}_i$ via a vector commitment with succinct opening [CF13], and instead of providing all the $\mathbf{t}_i$ to NISC_j, we will let the receiver only input a small number of the $\mathbf{t}_i$ (needed to correct the receiver's errors) together with a succinct opening of the vector commitment. In the same manner, we will use vector commitments with succinct opening to bind to the vectors $\mathbf{d}_{i,j}$ given in the CRS and to the ciphertexts $c_{i,j}$.

We will now describe the NISC functionality NISC_j for column j.

The functionality for column j takes as input from the receiver the LPN secret s, the error vector $\mathbf{e}_j$ (represented by a succinct list of length at most $2\epsilon m$). Furthermore, for every index i in the support of $\mathbf{e}_j$ the receiver inputs $\mathbf{t}_i$ together with a succinct opening of the corresponding vector commitment.

The sender provides as input $\mathbf{r}_j$, and the error vectors $\mathbf{f}_{j,0}$ and $\mathbf{f}_{j,1}$ (each represented by a succinct list of length $2\delta m$), as well as the key K_j. For every index i in the support of either $\mathbf{f}_{j,0}$ or $\mathbf{f}_{j,1}$, the sender also inputs $c_{i,j}$ (together with a succinct opening of the corresponding vector commitment) and $\mathbf{d}_{i,j}$ (also with an opening of the corresponding vector commitment).

The functionality performs the following computations. It punctures K_j at the support of $\mathbf{e}_j$ yielding a key $K_j^{\odot}$, computes offsets $\gamma_{i,j} = \mathsf{PRF}_{K_j}(i) \oplus \langle \mathbf{t}_i, \mathbf{r}_j \rangle$ for i in the support of $\mathbf{e}_j$, as well as $h_j = \mathsf{H}(\mathsf{hk}, \mathbf{r}_j)$. For every index i in the support of either $\mathbf{f}_{j,0}$ or $\mathbf{f}_{j,1}$, the functionality decrypts $c_{i,j}$ via $b_{i,j} \leftarrow \langle s, \mathbf{d}_{i,j} \rangle \oplus e_{i,j} \oplus c_{i,j}$. If i is in the support of $\mathbf{f}_{j,b_{i,j}}$, it sets $\beta_{i,j} = 1$ and otherwise $\beta_{i,j} = 0$. Note that the $(\beta_{i,j})_{i \in [m]}$ can be described by a succinct list of length at most $4\delta m$.

The functionality then outputs $K_j^{\odot}$, $(\gamma_{i,j})_{i \in [m]}$, h_j as well as $(\beta_{i,j})_{i \in [m]}$ to the receiver.

Now the receiver proceeds as follows. For every i not in the support of $\mathbf{e}_j$, he uses the punctured key $K_j^{\odot}$ to remove the mask $\mathsf{PRF}_{K_j}(i)$ from $w_{i,j,b_{i,j}}$ and use $\beta_{i,j}$ to correct a potential error. Specifically, he computes

$$m'_{i,j,b_{i,j}} \leftarrow \mathsf{Dec}'(\mathsf{Eval}((c_{i,j}, \mathbf{d}_{i,j}), \mathsf{td}_{i,0}, \ldots, \mathsf{td}_{i,n}), h_j) \oplus w_{i,j,b_{i,j}} \oplus \mathsf{PRF}_{K_j^{\odot}}(i) \oplus \beta_{i,j}.$$

For every i in the support of $\mathbf{e}_j$, the receiver can correct $w_{i,j,b_{i,j}}$ via $\gamma_{i,j}$, i.e. he computes

$$m'_{i,j,b_{i,j}} \leftarrow \mathsf{Dec}'(\mathsf{Eval}((c_{i,j}, \mathbf{d}_{i,j}), \mathsf{td}_{i,0}, \ldots, \mathsf{td}_{i,n}), h_j) \oplus w_{i,j,b_{i,j}} \oplus \gamma_{i,j} \oplus \beta_{i,j}.$$

We will first discuss correctness of the modified scheme. First note that by construction it holds that $\beta_{i,j} = f_{i,j,b_{i,j}}$. Hence, for i not in the support of $\mathbf{e}_j$ it holds that

$$\begin{aligned} m'_{i,j,b_{i,j}} &= \mathsf{Dec}'(\mathsf{Eval}((c_{i,j}, \mathbf{d}_{i,j}), \mathsf{td}_{i,0}, \ldots, \mathsf{td}_{i,n}), h_j) \oplus w_{i,j,b_{i,j}} \oplus \mathsf{PRF}_{K_j^{\odot}}(i) \oplus \beta_{i,j} \\ &= m_{i,j,b_{i,j}} \oplus f_{i,j,b_{i,j}} \oplus \beta_{i,j} \\ &= m_{i,j,b_{i,j}} \end{aligned}$$

as desired. Here, the first equality follows from

$$w_{i,j,b_{i,j}} = \mathsf{Enc}(\mathsf{ek}'_{i,j}, \mathbf{r}_j) \oplus b_{i,j} \langle \mathbf{t}_i, \mathbf{r}_j \rangle \oplus f_{i,j,b_{i,j}} \oplus m_{i,j,b_{i,j}} \oplus \mathsf{PRF}_{K_j}(i).$$

For i in the support of $\mathbf{e}_j$, it holds that

$$\begin{aligned} m'_{i,j,b_{i,j}} &= \mathsf{Dec}'(\mathsf{Eval}((c_{i,j}, \mathbf{d}_{i,j}), \mathsf{td}_{i,0}, \ldots, \mathsf{td}_{i,n}), h_j) \oplus w_{i,j,b_{i,j}} \oplus \gamma_{i,j} \oplus \beta_{i,j} \\ &= e_{i,j} \langle \mathbf{t}_i, \mathbf{r}_j \rangle \oplus f_{i,j,b_{i,j}} \oplus m_{i,j,b_{i,j}} \oplus \mathsf{PRF}_{K_j}(i) \oplus \gamma_{i,j} \oplus \beta_{i,j} \\ &= \langle \mathbf{t}_i, \mathbf{r}_j \rangle \oplus \oplus m_{i,j,b_{i,j}} \oplus \mathsf{PRF}_{K_j}(i) \oplus \gamma_{i,j} \\ &= \langle \mathbf{t}_i, \mathbf{r}_j \rangle \oplus \oplus m_{i,j,b_{i,j}} \langle \mathbf{t}_i, \mathbf{r}_j \rangle \\ &= m_{i,j,b_{i,j}}, \end{aligned}$$

where the second equality follows from $\beta_{i,j} = f_{i,j,b_{i,j}}$, the third equality from $e_{i,j} = 1$ for i in the support of $\mathbf{e}_j$, and the fourth equality from the definition of $\gamma_{i,j}$. Hence the modified scheme is correct.

The security proof proceeds along similar lines as in [BBDP22], in that we can replace the PRF outputs $\mathsf{PRF}_{K_j}(i)$ at punctured points i by uniformly random values to make the programming via the $\gamma_{i,j}$ undetectable.

Concerning the additional communication overhead induced by these modifications, note the following. For every $j \in [\ell]$, $\mathbf{NISC}_j$ takes inputs of size sub-linear in m and only performs a few local computations. Hence, using a NISC protocol such as [IKO+11] (which can be instantiated using any 2-message OT protocol), we obtain for each NISC_j a communication overhead sub-linear in m, hence by choosing m and ℓ as sufficiently large polynomials, we achieve amortized upload complexity approaching 1 bit, and amortized download complexity approaching 2 bits.

This concludes the outline.

2.7 Discussion

A natural question arising considering the results of our work is whether trapdoor hash function TDH in our protocol can be instantiated from assumptions other than QR, e.g. via the DDH-based trapdoor hash function given in [DGI+19]. A point of notice is that our construction makes critical use of perfect correctness properties of the underlying TDH, as it allows us to simulate the views of malicious parties. The DDH-based construction in [DGI+19] has a correctness error, and mitigating this correctness error requires the "assistance" of the evaluator. This is highly problematic for the case of malicious evaluators, since we cannot rely on tools such as NIZK proofs to enforce honest behaviour by the evaluator as this would ruin the rate of the TDH. Consequently, trying to instantiate this blueprint with TDH that have an additional correctness error is beyond the scope of this work.

3 Preliminaries

The acronym PPT denotes "probabilistic polynomial time". Throughout this work, λ denotes the security parameter. By $\mathsf{negl}(\lambda)$, we denote a negligible function in λ, that is, a function that vanishes faster than any inverse polynomial in λ.

Let $n \in \mathbb{N}$. Then, $[n]$ denotes the set $\{1, \dots, n\}$. If $\mathcal{A}$ is an algorithm, we denote by $y \leftarrow \mathcal{A}(x)$ the output y after running $\mathcal{A}$ on input x. If S is a (finite) set, we denote by $x \leftarrow\!\!\$\ S$ the experiment of sampling uniformly at random an element x from S. If D is a distribution over S, we denote by $x \leftarrow\!\!\$\ D$ the element x sampled from S according to D.

For two probability distributions X, Y, we use the notation $X \approx_s Y$ to state that the distributions are statistically indistinguishable.

We denote by $\mathsf{Supp}(\mathbf{u})$ the support of $\mathbf{u}$, that is, the set of indices where $\mathbf{u}$ is different from 0.[4] For $S \subseteq [n]$, $\mathbf{u}_S$ denotes the vector $(\mathbf{u}_i)_{i \in S}$. Finally, $\mathbf{u}^T$ denotes the transpose of $\mathbf{u}$.

3.1 Hardness Assumptions

The hardness assumptions used in this work are the learning parity with errors (LPN) and the quadratic residuosity (QR) assumptions.

3.1.1 Learning Parity with Noise

Informally, the LPN assumption states that it is hard to find a solution for a noisy system of linear equations over $\mathbb{Z}_2$. We now give the precise definition of the assumption.

Definition 1 (LPN assumption). *Let $n, m, t \in \mathbb{N}$ such that $n \in \mathsf{poly}(\lambda)$ and let $\chi_{m,t}$ be uniform distribution over the set of error vectors of size m and hamming weight t. The* Learning Parity with Noise *(LPN)*

[4]If there is only one index different from zero, $\mathsf{Supp}(\mathbf{u})$ denotes this index.

assumption $\mathsf{LPN}(n, m, t)$ *holds if for any PPT adversary* $\mathcal{A}$ *we have that*

$$\left| \begin{array}{c} \Pr\left[1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{s}\mathbf{A} + \mathbf{e}) : \begin{array}{c} \mathbf{A} \leftarrow\!\!\$\ \{0,1\}^{n \times m} \\ \mathbf{s} \leftarrow\!\!\$\ \{0,1\}^{n} \\ \mathbf{e} \leftarrow\!\!\$\ \chi_{m,t} \end{array}\right] - \\ \Pr\left[1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{y}) : \begin{array}{c} \mathbf{A} \leftarrow\!\!\$\ \{0,1\}^{n \times m} \\ \mathbf{y} \leftarrow\!\!\$\ \{0,1\}^{m} \end{array}\right] \end{array} \right| \leq \mathsf{negl}(\lambda)$$

where $\rho = m/t$ *(*ρ *is called the noise rate).*

In this work, we assume that the noise rate ρ is $m^{1-\varepsilon}$, where $m = \mathsf{poly}(n)$, for any constant $\varepsilon > 0$. The LPN assumption is believed to be hard for that noise rate (see e.g. [BCG+19a] and references therein).

3.1.2 Quadratic Residuosity Assumption

In the following, let N is a Blum integer if $N = p \cdot q$ for some primes p and q such that $p \pmod 4 = q \pmod 4 = 3$. Moreover, we say p and q are safe primes if $p = 2p' + 1$ and $q = 2q' + 1$ for some prime numbers p', q'. We denote by $\mathbb{J}_N$ the multiplicative group of the elements in $\mathbb{Z}_N^*$ with Jacobi symbol $+1$ and by $\mathbb{QR}_N$ the multiplicative group of quadratic residues modulo N with generator g. Note that $\mathbb{QR}_N$ is a subgroup of $\mathbb{J}_N$ and they have order $\frac{\varphi(N)}{4}$ and $\frac{\varphi(N)}{2}$, respectively, where $\varphi(\cdot)$ is Euler's totient function. It is useful to write $\mathbb{J}_N \simeq \mathbb{H} \times \mathbb{QR}_N$, where $\mathbb{H}$ is the multiplicative group $(\pm 1, \cdot)$ of order 2. Note that if N is a Blum integer then $\gcd\left(2, \frac{\varphi(N)}{4}\right) = 1$ and $-1 \in \mathbb{J}_N \setminus \mathbb{QR}_N$. We recall the quadratic residuosity (QR) assumption [BG10].

Definition 2 (Quadratic Residuosity Assumption)**.** *Let* N *be a uniformly sampled Blum integer and let* $\mathbb{QR}_N$ *be the multiplicative group of quadratic residues modulo* N *with generator* g*. We say the QR assumption holds with respect to* $\mathbb{QR}_N$ *if for any PPT adversary* $\mathcal{A}$

$$|\Pr[1 \leftarrow \mathcal{A}(N, g, a)] - \Pr[1 \leftarrow \mathcal{A}(N, g, (-1) \cdot a)]| \leq \mathsf{negl}(\lambda)$$

where $a \leftarrow\!\!\$\ \mathbb{QR}_N$*.*

3.2 Cryptographic Primitives

Here we define the relevant cryptographic primitives that we use in this work.

3.2.1 PRGs and Puncturable PRFs

We recall the definition of pseudorandom generators.

Definition 3 (Pseudorandom Generators)**.** *Let* $\alpha = \alpha(\lambda)$ *and* $\beta = \beta(\lambda)$ *be two polynomials. A pseudorandom generator* $\mathsf{PRG} : \{0,1\}^\alpha \to \{0,1\}^\beta$ *is a function such that for all PPT adversaries* $\mathcal{A}$ *we have that*

$$\left| \Pr\left[1 \leftarrow \mathcal{A}(\nu) : \begin{array}{c} s \leftarrow\!\!\$\ \{0,1\}^\alpha \\ \nu \leftarrow \mathsf{PRG}(s) \end{array}\right] - \Pr\left[1 \leftarrow \mathcal{A}(\nu) : \nu \leftarrow\!\!\$\ \{0,1\}^\beta\right] \right| \leq \mathsf{negl}(\lambda).$$

A pseudorandom function (PRF) is a pair of functions $\mathsf{KeyGen}, \mathsf{Eval}$ where $\mathsf{Eval} : \mathcal{K} \times \{0,1\}^\alpha \to \{0,1\}^\beta$ (for some $\alpha, \beta = \mathsf{poly}(\lambda)$) is computed by a deterministic polynomial time algorithm: On input $(\mathsf{K}, \mathbf{x}) \in \mathcal{K} \times \{0,1\}^\alpha$ the algorithm outputs $\mathsf{Eval}(\mathsf{K}, \mathbf{x}) = \mathbf{y} \in \{0,1\}^\beta$. In terms of security, the value $\mathbf{y}$ is pseudorandom. This is captured by a game where a PPT adversary gets either oracle access to $\mathsf{Eval}(\mathsf{K}, \mathbf{x})$ or to a random function.

Puncturable pseudorandom functions (PPRFs) [BW13, KPTZ13, BGI14] are a special case of PRFs where a punctured key allows one to evaluate the PRF at all points except one.

Definition 4 (Puncturable PRF)**.** *Let $\alpha = \alpha(\lambda)$ and $\beta = \beta(\lambda)$ be two polynomials. A puncturable PRF (PPRF) scheme $\mathsf{PPRF}_{\alpha,\beta} = \mathsf{PPRF}$ is composed by the following algorithms:*

- $\mathsf{KeyGen}(1^\lambda)$ *takes as input a security parameter λ. It outputs a key K.*
- $\mathsf{Eval}(\mathsf{K}, \mathbf{x})$ *takes as input a key K and $x \in \{0,1\}^\alpha$. It outputs $\mathbf{y} \in \{0,1\}^\beta$.*
- $\mathsf{Punct}(\mathsf{K}, S)$ *takes as input a key K and a subset $S \subseteq \{0,1\}^\alpha$. It outputs a punctured key K_S.*
- $\mathsf{EvalPunct}(\mathsf{K}_S, \mathbf{x})$ *takes as input a punctured key K_S and $\mathbf{x} \in \{0,1\}^\alpha$. It outputs $\mathbf{y} \in \{0,1\}^\beta$.*

Definition 5 (Correctness)**.** *A PPRF scheme PPRF is said to be correct if for all $\lambda \in \mathbb{N}$, for all $S \subseteq (\{0,1\}^\alpha)^t$ (for $t = \mathsf{poly}(\lambda)$), all $\mathbf{x} \notin S$ we have that*

$$\Pr\left[\mathsf{Eval}(\mathsf{K}, \mathbf{x}) = \mathsf{EvalPunct}(\mathsf{K}_S, \mathbf{x}) : \begin{array}{c} \mathsf{K} \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{K}_S \leftarrow \mathsf{Punct}(\mathsf{K}, S) \end{array}\right] = 1.$$

We define $T \leftarrow \mathsf{Eval}(\mathsf{K}, S)$ to be the set of points $\mathbf{y} \leftarrow \mathsf{Eval}(\mathsf{K}, \mathbf{x})$ where $\mathbf{x} \in S$.

Definition 6 (Pseudorandomness)**.** *A PPRF scheme PPRF is said to be pseudorandom at punctured points if for all $\lambda \in \mathbb{N}$, all PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have that*

$$\left| \begin{array}{c} \Pr\left[1 \leftarrow \mathcal{A}_2(\mathsf{K}_S, S, T, \mathsf{aux}) : \begin{array}{c} (S, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda);\ \mathsf{K} \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{K}_S \leftarrow \mathsf{Punct}(\mathsf{K}, S);\ T \leftarrow \mathsf{Eval}(\mathsf{K}, S) \end{array}\right] - \\ \Pr\left[1 \leftarrow \mathcal{A}_2(\mathsf{K}_S, S, T, \mathsf{aux}) : \begin{array}{c} (S, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda);\ \mathsf{K} \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{K}_S \leftarrow \mathsf{Punct}(\mathsf{K}, S);\ T \leftarrow_\$ \{0,1\}^{\beta|S|} \end{array}\right] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

PPRFs can be built solely based on any length-doubly pseudorandom generators (PRG)[5] via (a variant of) the tree-based construction of [GGM86].

3.2.2 Functional Laconic Oblivious Transfer

In this section, we present a variant of laconic oblivious transfer [CDG+17], which we call functional laconic oblivious tranfer (FLOT). Here, the receiver hashes a database $\mathbf{D}$ composed of several blocks $(\mathbf{D}_1, \ldots, \mathbf{D}_\nu)$. The sender can compute a ciphertext with respect to a position $i \in [\nu]$ encrypting a value $\mathbf{y}$ which allows the receiver to obtain $F(\mathbf{D}_i, y)$.

Definition 7. *Let F be any polynomial-time function. A functional laconic oblivious transfer (FLOT) scheme FLOT for a function $F : \{0,1\}^L \times \{0,1\}^\ell \to \{0,1\}$ is composed by the following algorithms:*

- $\mathsf{Setup}(1^\lambda)$ *takes as input a security parameter λ. It outputs a common reference string crs.*
- $\mathsf{H}(\mathsf{crs}, \mathbf{D} \in \{0,1\}^{\nu \cdot L})$ *takes as input a common reference string crs a database $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_\nu)$ where each block $\mathbf{D}_i \in \{0,1\}^L$. It outputs a hash value h.*
- $\mathsf{Enc}(\mathsf{crs}, h, i \in [\nu], \mathbf{y} \in \{0,1\}^\ell)$ *takes as input a common reference string crs, a hash value h, a position $i \in [\nu]$ and a vector $\mathbf{y} \in \{0,1\}^\ell$. It outputs a ciphertext lotct.*
- $\mathsf{Dec}(\mathsf{crs}, \mathbf{D}, \mathsf{lotct}, i)$ *takes as input a common reference string crs, a database $\mathbf{D}$, a ciphertext lotct and a position i. It outputs a value $z \in \{0,1\}$.*

Definition 8 (Correctness)**.** *A FLOT is said to be correct if for all $\mathbf{D} \in \{0,1\}^{\nu \cdot L}$, $i \in [\nu]$, $\mathbf{y} \in \{0,1\}^\ell$ $\lambda \in \mathbb{N}$ we have that*

$$\Pr\left[z = F(\mathbf{D}_i, \mathbf{y}) : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ h \leftarrow \mathsf{H}(\mathsf{crs}, \mathbf{D}) \\ \mathsf{lotct} \leftarrow \mathsf{Enc}(\mathsf{crs}, h, i, \mathbf{y}) \\ z \leftarrow \mathsf{Dec}(\mathsf{crs}, \mathbf{D}, \mathsf{lotct}, i) \end{array}\right] = 1.$$

[5]Which in turn, can be based on LWE, DDH or QR assumptions.

Definition 9 (Sender security)**.** *A FLOT is said to be sender secure if for all PPT advrsaries $\mathcal{A}$ there exists a simulator* Sim *such that for all databases $\mathbf{D} \in \{0,1\}^{\nu \cdot L}$, any position $i \in [\nu]$, any vector $\mathbf{y} \in \{0,1\}^{\ell}$ we have that*

$$\left| \begin{array}{l} \Pr\left[1 \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{lotct}) : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ h \leftarrow \mathsf{H}(\mathsf{crs}, \mathbf{D}) \\ \mathsf{lotct} \leftarrow \mathsf{Enc}(\mathsf{crs}, h, i, \mathbf{y}) \end{array} \right] - \\ \Pr\left[1 \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{lotct}) : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ h \leftarrow \mathsf{H}(\mathsf{crs}, \mathbf{D}) \\ \mathsf{lotct} \leftarrow \mathsf{Sim}(\mathsf{crs}, h, i, F(\mathbf{D}_i, \mathbf{y})) \end{array} \right] \end{array} \right| = \frac{1}{2} + \mathsf{negl}(\lambda)$$

Definition 10 (Compactness)**.** *A FLOT is said to be compact if $|\mathsf{crs}| = |h| = \mathsf{poly}(\lambda)$, and if the running time of* Enc *is bounded by $\mathsf{poly}(F, |\mathbf{y}|, |\mathbf{D}_i|, \lambda)$.*

Instantiation. In Appendix A, we show how to build this flavor of laconic oblivious transfer from the quadratic residuosity assumption.

3.2.3 Vector Commitments with Local Openings

Vector commitments [CF13] are commitment schemes that have the following properties: the commitment is short, when compared to the message, and we can locally open just a position of the committed message with small communication. They can be instantiated using Merkle trees.

Definition 11 (Vector commitment)**.** *A vector commitment* VC *is composed by the following algorithms:*

- $\mathsf{Setup}(1^\lambda)$ *takes as input a security parameter λ. It outputs a common reference string* crs.
- $\mathsf{Com}(\mathsf{crs}, \mathbf{D} \in \{0,1\}^\nu)$ *takes as input a common reference string* crs *a database $\mathbf{D} = (d_1, \dots, d_\nu)$. It outputs a commitment* com *and a state* st.
- $\mathsf{Open}(\mathsf{crs}, \mathsf{com}, \mathsf{st}, i)$ *takes as input a common reference string* crs*, a commitment* com*, a state* st *and a position $i \in [\nu]$. It outputs an opening δ.*
- $\mathsf{Verify}(\mathsf{crs}, \mathsf{com}, d_i, i, \delta)$ *takes as input a common reference string* crs*, a commitment* com*, a bit $d_i \in \{0,1\}$, a position $i \in [\nu]$ and an opening δ. It outputs a value $z \in \{0,1\}$.*

A VC is compact if $|\mathsf{crs}| = |\mathsf{com}| = |\delta| = \mathsf{poly}(\lambda)$. In particular, they do not depend on the size of $\mathbf{D} \in \{0,1\}^\nu$.

Definition 12 (Correctness)**.** *A VC is said to. be correct if for all databases $\mathbf{D} = (d_1, \dots, d_\nu) \in \{0,1\}^\nu$ and all indices $i \in [\nu]$ we have that*

$$\Pr\left[1 \leftarrow \mathsf{Verify}(\mathsf{crs}, \mathsf{com}, d_i, i, \delta) : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{com}, \mathsf{st}) \leftarrow \mathsf{Com}(\mathsf{crs}, \mathbf{D} \in \{0,1\}^\nu) \\ \delta \leftarrow \mathsf{Open}(\mathsf{crs}, \mathsf{com}, \mathsf{st}, i) \end{array} \right] = 1.$$

Definition 13 (Position binding)**.** *A VC is said to be position binding if for all $i \in [\nu]$ and PPT all adversaries $\mathcal{A}$ we have that*

$$\Pr\left[\begin{array}{c} 1 \leftarrow \mathsf{Verify}(\mathsf{crs}, \mathsf{com}, m, i, \delta) \wedge \\ 1 \leftarrow \mathsf{Verify}(\mathsf{crs}, \mathsf{com}, m', i, \delta') \wedge m \neq m' \end{array} \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{com}, m, m', i, \delta, \delta') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Instantiation. Vector commitments can be instantiated using Merkle trees. Hence, we can instantiate the VC from the QR assumption using a appropriate collision-resistant hash function from the QR assumption [BG10].

3.3 UC-Security and Ideal Functionalities

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

Definition 14. *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* $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{E}}$ *and* $\mathsf{REAL}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{E}}$ *are computationally indistinguishable.*

3.3.1 NISC Functionality

Let $\mathcal{C} : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ be a circuit. We define the ideal functionality the NISC functionality that allows two parties to jointly compute $\mathcal{C}(x, y)$.

NISC functionality $\mathcal{F}_{\mathcal{C}}$ for $\mathcal{C}$. This implements the following functionality.

- **Receiver's input:** $x \in \mathcal{X}$.
- **Sender's input:** $y \in \mathcal{Y}$.
- **Receiver's output:** $\mathcal{C}(x, y)$.

Given a circuit $\mathcal{C}$ the functionality can be implemented in two rounds using a non-interactive secure computation scheme (NISC).

Lemma 1 ([IKO+11])**.** *Given a circuit* $\mathcal{C}$*, the functionality* $\mathcal{F}_{\mathcal{C}}$ *can be implemented in two-rounds by a NISC scheme. Additionally, a NISC scheme can be implemented using only a two-round OT (such as the one from [PVW08]). The total communication of the protocol is* $|\mathcal{C}| \cdot \mathsf{poly}(\lambda)$.

3.3.2 OT Functionality

We define the OT ideal functionality as in [PVW08].

OT functionality. A (batch) OT scheme should implement the following functionality.

- **Receiver's input:** A string of bits $\mathbf{b} = (b_1, \ldots, b_k) \in \{0, 1\}^k$.
- **Sender's input:** Pairs of messages $(\mathbf{m}_0, \mathbf{m}_1) \in (\{0, 1\}^k)^2$, where $\mathbf{m}_0 = (m_{0,1}, \ldots, m_{0,k})$ and $\mathbf{m}_1 = (m_{1,1}, \ldots, m_{1,k})$
- **Receiver's output:** The string $\mathbf{m}_{\mathbf{b}} = (m_{b_1,1}, \ldots, m_{b_k,k})$.

4 Key-Homomorphic Trapdoor Hash Function

We start by defining key-homomorphic trapdoor hash (KH-TDH). This primitive is similar to the one presented in [DGI+19] except that it allows for homomorphic operations over evaluation keys. Here, we define homomorphism just for linear functions which is enough for our application.

Definition 15 (Key-homomorphic trapdoor hash function)**.** *Let* $n \in \mathbb{N}$*. A key-homomorphic trapdoor hash function (KH-TDH) is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{H}, \mathsf{Enc}, \mathsf{Dec})$

- $\mathsf{Setup}(1^\lambda, L)$ *takes as input a security parameter and an integer* $L \in \mathbb{N}$*. It outputs a hash key* hk.

- $\mathsf{KeyGen}(\mathsf{hk}, \mathbf{z}, \mathbf{t})$ *takes as input a hash key* hk *and two vectors* $\mathbf{z} \in \{0,1\}^n$ *and* $\mathbf{t} \in \{0,1\}^L$*. It outputs a evaluation key* ek *and a trapdoor* td.
- $\mathsf{Eval}(\mathsf{ek}, (\mathbf{d}, c), \mathbf{t})$ *takes as input an evaluation key* ek*, a linear function* $(\mathbf{d}, c) \in \{0,1\}^n \times \{0,1\}$ *and a vector* $\mathbf{t} \in \{0,1\}^L$*. It outputs a new evaluation key* ek'.
- $\mathsf{H}(\mathsf{hk}, \mathbf{r})$ *takes as input a hash key* hk *and a vector* $\mathbf{r} \in \{0,1\}^L$*. It outputs a hash value* h*.*
- $\mathsf{Enc}(\mathsf{ek}', \mathbf{r})$ *takes as input an evaluation key* ek' *and a vector* $\mathbf{r} \in \{0,1\}^L$*. It outputs an encoding* $a \in \{0,1\}$*.*
- $\mathsf{Dec}(\mathsf{td}, (\mathbf{d}, c), h)$ *takes as input a trapdoor* td*, a linear function* $(\mathbf{d}, c) \in \{0,1\}^n \times \{0,1\}$ *and a hash value* h*. It outputs an encoding* $a' \in \{0,1\}$*.*

Correctness. For all integers $n, L \in \mathbb{N}$, all vectors $\mathbf{z} \in \{0,1\}^n$, $\mathbf{t} \in \{0,1\}^L$, $\mathbf{r} \in \{0,1\}^L$ and all linear functions $(\mathbf{d}, c) \in \{0,1\}^n \times \{0,1\}$ we have that

$$\Pr\left[a + a' = (\mathbf{z} \cdot \mathbf{d}^T + c) \cdot (\mathbf{t} \cdot \mathbf{r}^T) : \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, L) \\ (\mathsf{ek}, \mathsf{td}) \leftarrow \mathsf{KeyGen}(\mathsf{hk}, \mathbf{z}, \mathbf{t}) \\ \mathsf{ek}' \leftarrow \mathsf{Eval}(\mathsf{ek}, (\mathbf{d}, c), \mathbf{t}) \\ h \leftarrow \mathsf{H}(\mathsf{hk}, \mathbf{r}) \\ a \leftarrow \mathsf{Enc}(\mathsf{ek}', \mathbf{r}) \\ a' \leftarrow \mathsf{Dec}(\mathsf{td}, (\mathbf{d}, c), h) \end{array} \right] = 1$$

Receiver privacy. For all $\lambda \in \mathbb{N}$, all $n, L \in \mathbb{N}$, all vectors $\mathbf{z}_0, \mathbf{z}_1 \in \{0,1\}^n$ and $\mathbf{t} \in \{0,1\}^L$ and all PPT adversaries $\mathcal{A}$, we have that

$$\Pr\left[b \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{ek}) : \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, L) \\ b \leftarrow\!\!\$\ \{0,1\} \\ (\mathsf{ek}, \mathsf{td}) \leftarrow \mathsf{KeyGen}(\mathsf{hk}, \mathbf{z}_b, \mathbf{t}) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Sender security. For all $\lambda \in \mathbb{N}$, all $n, L \in \mathbb{N}$ such that $L = \omega(\lambda)$, all vectors $\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^L$ and all PPT adversaries $\mathcal{A}$, we have that

$$\Pr\left[b \leftarrow \mathcal{A}(\mathsf{hk}, h) : \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, n, L) \\ b \leftarrow\!\!\$\ \{0,1\} \\ h \leftarrow \mathsf{H}(\mathsf{hk}, \mathbf{r}) \text{ if } b = 0 \\ h \leftarrow \{0,1\}^\lambda \text{ if } b = 1 \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

4.1 Construction from QR

We recall the shrinking mechanism of [DGI+19] (formalized in [BBD+20]). Let a (packed) ciphertext $\mathsf{ct} = (g^r, (-1)^{b_1} h_1^r, \ldots, (-1)^{b_k} h_k^r) = (c_1, c_{2,1}, \ldots, c_{2,k})$ and let $<$ be an order over $\mathbb{J}_N$ (e.g., the lexicographic order). The shrinking mechanism of [DGI+19] simply outputs 0 if $c_{2,i} < -c_{2,i}$ and outputs 1 otherwise. We will denote this procedure by $\mathsf{Shrink}_{\mathsf{QR}} : \mathbb{J}_N \to \{0,1\}$. Note that this procedure is completely deterministic.

Lemma 2 (Linear homomorphism)**.** *For any* $x \in \mathbb{J}_N$ *we have that* $\mathsf{Shrink}_{\mathsf{QR}}(x \cdot (-1)) \mod 2 = \mathsf{Shrink}_{\mathsf{QR}}(x) + 1 \mod 2$

Proof. If $x < -x$ then $\mathsf{Shrink}_{\mathsf{QR}}(x) = 0$ and $\mathsf{Shrink}_{\mathsf{QR}}(x \cdot (-1)) = 1$. Then $\mathsf{Shrink}_{\mathsf{QR}}(x \cdot (-1)) = \mathsf{Shrink}_{\mathsf{QR}}(x) + 1$. The other case follows using a similar reasoning. □

In this section we present our key-homomorphic TDH from QR. The construction shares similarities with the one from [DGI+19]. The main difference is that we show keys are linearly homomorphic.

Construction 1. *Let* $L, n \in \mathsf{poly}(\lambda)$*. Let* $\mathsf{Shrink}_{\mathsf{QR}}$ *be the algorithm described above.*

$\mathsf{Setup}(1^\lambda, L)$:

- *Generate two safe prime numbers* P, Q *and compute* $N = P \cdot Q$. *Sample* $\mathbf{g} = (g_1, \ldots, g_L) \leftarrow\!\$\ \mathbb{QR}_N^L$.
- *Output* $\mathsf{hk} = (N, \mathbf{g})$.

$\mathsf{KeyGen}(\mathsf{hk}, \mathbf{z} \in \{0,1\}^n, \mathbf{t} \in \{0,1\}^L)$:

- *Parse* hk *as* $(N, \mathbf{g})$, $\mathbf{z} = (z_1, \ldots, z_n)$ *and* $\mathbf{t} = (t_1, \ldots, t_L)$.
- *Sample* $\mathbf{s} = (s_1, \ldots, s_n) \leftarrow\!\$\ [(N-1)/2]$.
- *For all* $i \in [n]$ *and all* $j \in [L]$ *compute* $y_{j,i} = g_j^{s_i}(-1)^{z_i \cdot t_j} \mod N$.
- *Output* $\mathsf{ek} = \{y_{j,i}\}_{i\in[n], j\in[L]}$ *and* $\mathsf{td} = (\mathbf{s}, \mathbf{t})$.

$\mathsf{Eval}(\mathsf{ek}, (\mathbf{d} \in \{0,1\}^n, c \in \{0,1\}), \mathbf{t})$:

- *Parse* ek *as* $\{y_{j,i}\}_{i\in[n], j\in[L]}$, $\mathbf{d} = (d_1, \ldots, d_n)$.
- *For all* $j \in [L]$ *compute* $w_j = \prod_{i=1}^n y_{j,i}^{d_i} \cdot (-1)^{c \cdot \mathbf{t}} \mod N$.
- *Output* $\mathsf{ek}' = \{w_j\}_{j\in[L]}$.

$\mathsf{H}(\mathsf{hk}, \mathbf{r} \in \{0,1\}^L)$:

- *Parse* hk *as* $(N, \mathbf{g})$ *where* $\mathbf{g} = (g_1, \ldots, g_L)$.
- *Output* $h = \prod_{j=1}^L g_j^{r_j} \mod N$.

$\mathsf{Enc}(\mathsf{ek}', \mathbf{r} \in \{0,1\}^L)$:

- *Parse* ek' *as* $\{w_j\}_{j\in[L]}$.
- *Output* $a = \mathsf{Shrink}_{\mathsf{QR}}(\prod_{j=1}^L w_j^{r_j} \mod N)$.

$\mathsf{Dec}(\mathsf{td}, (\mathbf{d} \in \{0,1\}^n, c \in \{0,1\}), h)$:

- *Parse* td *as* $(\mathbf{s}, \mathbf{t})$.
- *Compute* $s' = \mathbf{s} \cdot \mathbf{d}^T \in \mathbb{Z}_N$.
- *Output* $a' = \mathsf{Shrink}_{\mathsf{QR}}(h^{s'} \mod N)$.

We now analyze correctness of the scheme.

Lemma 3. *The scheme presented in Construction 1 is correct.*

Proof. First note that

$$\prod_{j=1}^L w_j^{r_j} \mod N = \prod_{j=1}^L \left(\prod_{i=1}^n y_{j,i}^{d_i} \cdot (-1)^{c\cdot\mathbf{t}} \right)^{r_j} = \prod_{j=1}^L \left(\prod_{i=1}^n \left(g_j^{s_i} (-1)^{z_i \cdot t_j} \right)^{d_i} \cdot (-1)^{c\cdot\mathbf{t}} \right)^{r_j}$$

$$= (-1)^{(\mathbf{z}\cdot\mathbf{d}^T + c)\cdot(\mathbf{t}\cdot\mathbf{r}^T)} \prod_{j=1}^L g_j^{(\mathbf{s}\cdot\mathbf{d}^T)\cdot r_j}.$$

On the other hand

$$h^{s'} \mod N = \left(\prod_{j=1}^{L} g_j^{r_j}\right)^{\mathbf{s}\cdot\mathbf{d}^T} \mod N.$$

Hence by the linear homomorphic correctness of $\mathsf{Shrink}_{\mathsf{QR}}$ (Lemma 2) we have that

$$\begin{aligned} a + a' \mod 2 &= \mathsf{Shrink}_{\mathsf{QR}}\left(\prod_{j=1}^{L} w_j^{r_j} \mod N\right) + \mathsf{Shrink}_{\mathsf{QR}}(h^{s'} \mod N) \\ &= (\mathbf{z}\cdot\mathbf{d}^T)\cdot(\mathbf{t}\cdot\mathbf{r}^T) \mod 2. \end{aligned}$$

□

Lemma 4 (Receiver security). *The scheme presented in Construction 1 is receiver secure assuming that the QR assumption holds.*

This is a direct consequence of the QR assumption.

Lemma 5 (Sender security). *The scheme presented in Construction 1 is sender secure assuming that* $L = \Omega(\mathsf{poly}(\lambda))$.

Let $q, L \in \mathbb{N}$ such that $L = \Omega(\lambda)$. To prove this lemma, recall that the leftover hash lemma states that for $\mathbf{s} \leftarrow\$ \{0,1\}^L$ and for $\mathbf{b} \leftarrow\$ \mathbb{Z}_q^L$ then

$$(\mathbf{b}, \mathbf{s}\cdot\mathbf{b}^T) \approx_s (\mathbf{b}, u)$$

where $U \leftarrow\$ \mathbb{Z}_q$.

Proof. Let $g_1, \ldots, g_L \leftarrow\$ \mathbb{QR}_N$, then there exists a $\mathbf{b} \in \mathbb{Z}_{\phi(N)/4}$ such that $g^{b_i} = g_i$. Then the leftover hash lemma states that $g^u \approx_s g^{\mathbf{r}\cdot\mathbf{b}^T}$ for $u \leftarrow\$ \mathbb{Z}_{\phi(N)/4}$. □

Batch KH-TDH. We also define a batch version of the algorithms described above. Let $\mu \in \mathbb{N}$. For a set of vectors $\mathbf{t}_1, \ldots, \mathbf{t}_\mu \in \{0,1\}^L$ we define $\mathsf{ek} = (\mathsf{ek}_1, \ldots, \mathsf{ek}_\mu) \leftarrow \mathsf{KeyGen}(\mathsf{hk}, \mathbf{z} \in \{0,1\}^n, \mathbf{t}_1, \ldots, \mathbf{t}_\mu)$ where $\mathsf{ek}_i \leftarrow \mathsf{KeyGen}(\mathsf{hk}, \mathbf{z} \in \{0,1\}^n, \mathbf{t}_i)$ for all $i \in [\mu]$. Additionally, we define $\mathsf{ek}' = (\mathsf{ek}'_1, \ldots, \mathsf{ek}'_\mu) \leftarrow \mathsf{Eval}(\mathsf{ek}, (\mathbf{D} \in \{0,1\}^{n\times\mu}, \mathbf{c} \in \{0,1\}^\mu))$ as $\mathsf{ek}'_i \leftarrow \mathsf{Eval}(\mathsf{ek}, (\mathbf{d}_i \in \{0,1\}^n, c_i \in \{0,1\}))$ where $\mathbf{d}_i$ is the i-th column of $\mathbf{D}$. All other algorithms are defined analogously.

Communication complexity. We now analyze the communication complexity of our construction. Let $n, L, \mu \in \mathbb{N}$ defined as above.

- $\mathsf{hk} = L \cdot \mathsf{poly}(\lambda)$.
- $|\mathsf{td}| = n \cdot L \cdot \mathsf{poly}(\lambda)$ (in the batch version it has size $\mu \cdot L \cdot n \cdot \mathsf{poly}(\lambda)$).
- $|\mathsf{ek}| = L \cdot n \cdot \mathsf{poly}(\lambda)$ (in the batch version it has size $\mu \cdot L \cdot n \cdot \mathsf{poly}(\lambda)$).
- $|h| = \mathsf{poly}(\lambda)$.
- $|a| = |a'| = 1$ (in the batch version they have size μ).

Local Decryption. We define an additional property for our KH-TDH called local decryption. A KH-TDH is local decryptable if there exists an algorithm LocDec such that for all $\lambda \in \mathbb{N}$ we have that

$$\Pr\left[\mathbf{z}\cdot\mathbf{d}^T + c \leftarrow \mathsf{LocDec}(\mathsf{td},\mathbf{d},\mathsf{ek}') : \begin{array}{r} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, L) \\ (\mathsf{ek},\mathsf{td}) \leftarrow \mathsf{KeyGen}(\mathsf{hk},\mathbf{z},\mathbf{t}) \\ \mathsf{ek}' \leftarrow \mathsf{Eval}(\mathsf{ek},(\mathbf{d},c)) \end{array}\right] = 1.$$

In this batch version, given a batch of encoding keys $\mathsf{ek}' = (\mathsf{ek}'_1,\dots,\mathsf{ek}'_\mu)$ (obtained by evaluating $\mathbf{D} \in \{0,1\}^{n\times\mu}$ instead of a single vector $\mathbf{d}$), the LocDec algorithm can decrypt only one of ek'_i given the i-th column of $\mathbf{D}$.

It is easy to see that Construction 1 fulfills this definition. We explicitly present the LocDec algorithm.

$\mathsf{LocDec}(\mathsf{td},\mathbf{d},\mathsf{ek}')$:

- Parse ek' as $\{w_j\}_{j\in[L]}$ and td as $(\mathbf{s},\mathbf{t})$.
- Compute $s' = \mathbf{s}\cdot\mathbf{d}^T$.
- For all $j \in [L]$ compute $a_j = w_j / g_j^{s'} \mod N$.
- For all $i \in \mathsf{Supp}(\mathbf{t})$, if $a_i = 1$, output 0. Else if $a_i = -1$, output 1.

Upon evaluation $\mathsf{Eval}(\mathsf{ek},(\mathbf{d},c))$ we obtain the encoding key $\mathsf{ek}' = (g_1^{r'}\cdot(-1)^{(\mathbf{z}\cdot\mathbf{d}^T+c)\cdot t_1},\dots,g_L^{r'}\cdot(-1)^{(\mathbf{z}\cdot\mathbf{d}^T+c)\cdot t_L})$ where $r' = \mathbf{s}\cdot\mathbf{d}^T$. Correctness of LocDec follows easily.

Moreover, note that, since $|\mathsf{td}| = n\cdot L\cdot\mathsf{poly}(\lambda)$, the algorithm described above can be computed by a circuit of size $n\cdot L\cdot\mathsf{poly}(\lambda)$.

5 Composable Oblivious Transfer with Optimal Rate

In this section we present our optimal-rate OT scheme that achieves UC-security against malicous adversaries under the QR and LPN assumptions. Before we present our OT construction, we enumerate the necessary ingredients as well as an auxiliary ideal functionality. Additionally, we show that this ideal functionality can be implemented with sublinear communication (with respect to the total communication of the OT protocol).

5.1 Ingredients.

For our construction we need the following ingredients

- Let $L, n, \mu, t_1, t_2 \in \mathbb{N}$ such that $\nu = \sqrt{\mu}$.
- A key-homomorphic TDH scheme $\mathsf{TDH} = (\mathsf{Setup},\mathsf{KeyGen},\mathsf{Eval},\mathsf{H},\mathsf{Enc},\mathsf{LocDec},\mathsf{Dec})$.
- A puncturable PRF $\mathsf{PRF} = (\mathsf{KeyGen},\mathsf{Eval},\mathsf{Puncture},\mathsf{EvalPunct})$ such that $\mathsf{PRF.Eval} : ([\nu]\times[\nu]\times\{0,1\}) \to \{0,1\}$.
- A PRG $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\cdot\mu}$.
- A functional $\mathsf{LOT} = (\mathsf{Setup},\mathsf{H},\mathsf{Enc},\mathsf{Dec})$ for the function $F_{\mathbf{r}_i,\mathsf{K},b}(\mathbf{t}_j) = F(\mathbf{t}_j,(\mathbf{r}_i,i,\mathsf{K},b)) = \mathsf{PRF.Eval}(\mathsf{K},(i,j,b)) + (\mathbf{r}_i\cdot\mathbf{t}_j^T) \mod 2$.
- A vector commitment $\mathsf{VC} = (\mathsf{Com},\mathsf{Open},\mathsf{Verify})$ with local openings.

Auxiliary ideal functionality. We present an auxiliary ideal functionality that we will use in our OT protocol. This functionality implements four procedures: i) It checks if the receiver's TDH message is well-formed. ii) It checks if the sender's TDH hash is well-formed. iii) It corrects the receiver's LPN errors. And iv) it corrects the sender's LPN errors.

For technical reasons, we define an ideal functionality that implements these four procedures as it becomes easier to guarantee consistency of inputs for different procecures. Additionally, it gives the simulator enough power to extract the receiver's and sender's inputs that will allow for the simulation to go through in the security proof.

$\mathcal{G}_{\mathsf{aux}}$ ideal functionality. Consider the ideal functionality $\mathcal{G}_{\mathsf{aux}}$ such that:

Receiver's input. An LPN secret $\mathbf{s} \in \{0,1\}^n$, random coins $r \in \{0,1\}^\lambda$, a set of indices $S = \mathsf{Supp}(\mathbf{e})$, vectors $\mathbf{t}_1, \dots, \mathbf{t}_\nu \in \{0,1\}^L$, a hash key hk, a LOT hash h_{LOT} and a VC commitments com and com'.

Sender's input. An encoding key ek, a PRG seed seed, hash values $h_1, \dots, h_\nu$, vectors $\mathbf{r}_1, \dots, \mathbf{r}_\nu \in \{0,1\}^L$ and $\mathbf{t}'_1, \dots, \mathbf{t}'_\nu \in \{0,1\}^L$, a hash key hk', two support sets $T_0 = \mathsf{Supp}(\mathbf{f}_0), T_1 = \mathsf{Supp}(\mathbf{f}_1)$, vectors $\{\mathbf{d}_{i,j}\}_{(i,j)\in T_0\cup T_1}$, uncompressed ciphertexts $\{\gamma_{i,j}\}_{(i,j)\in T_0\cup T_1}$, openings $\{\delta_{i,j}\}_{i,j\in T_0\cup T_1}$ and $\{\phi_{i,j}\}_{i,j\in T_0\cup T_1}$ and a LOT hash h_{LOT}.

The functionality $\mathcal{G}$ implements the following functions:

F_{Cons} : This function checks consistency of inputs from the receiver and the sender.

- If $\mathsf{hk} \neq \mathsf{hk}'$ or $\mathbf{t}_1, \dots, \mathbf{t}_\nu \neq \mathbf{t}'_1, \dots, \mathbf{t}'_\nu$ or $h_{\mathsf{LOT}} \neq h'_{\mathsf{LOT}}$ abort.

F_{CDS}**:** This function checks if the receiver's TDH messsage is well-formed.

- Take as input $\mathbf{s} \in \{0,1\}^n$, $r \in \{0,1\}^\lambda$, $\mathbf{t}_1, \dots, \mathbf{t}_\nu \in \{0,1\}^L$ from the receiver. Take as input ek and seed from the sender.
- Parse $\mathsf{ek} = (\mathsf{ek}_1, \dots, \mathsf{ek}_\nu)$.
- If for all $i \in [\nu]$ $\mathsf{ek}_i \leftarrow \mathsf{TDH.KeyGen}(\mathsf{hk}, \mathbf{s}, \mathbf{t}_i : r)$ return seed to the receiver.
- Else abort.

F_{DVNIZK} : This function checks if the sender's message is well-formed.

- Take as input $h_1, \dots, h_\nu$ and $\mathbf{r}, \dots, \mathbf{r}_\nu$ from the sender.
- If for all $i \in [\nu]$ $h_i \leftarrow \mathsf{TDH.H}(\mathsf{hk}, \mathbf{r}_i)$ return $h_1, \dots, h_\nu$ to the receiver. Else abort.

F_{RecErr} : This function corrects the errors introduced by the receiver.

- Take as input a set of indices $S = \mathsf{Supp}(\mathbf{e})$ from the receiver. Take as input a PRF key K, a LOT hash h'_{LOT} and vectors $\mathbf{r}_1, \dots, \mathbf{r}_\nu$ from the sender.
- Compute $\mathsf{K}^* \leftarrow \mathsf{PRF.Puncture}(\mathsf{K}, \{(i,j,b) : (i,j) \in S, b \in \{0,1\}\})$.
- For all $(i,j) \in S$ compute $\mathsf{lotct}_{0,i,j} \leftarrow \mathsf{LOT.Enc}(\mathsf{crs}, h_{\mathsf{LOT}}, j, (\mathsf{K}, i, 0))$ and $\mathsf{lotct}_{1,i,j} \leftarrow \mathsf{LOT.Enc}(\mathsf{crs}, h_{\mathsf{LOT}}, j, (\mathsf{K}, i, 1))$.
- Output $(h_{\mathsf{LOT}}, \mathsf{K}^*, \{\mathsf{lotct}_{b,i,j}\}_{\beta\in\{0,1\},(i,j)\in S}$ to the receiver.

F_{SendErr} : This function corrects the errors introduced by the sender.

- Take as input $S = \mathsf{Supp}(\mathbf{e})$, td, $\mathbf{s} \in \{0,1\}^n$, com and com' from the receiver. Take as input a support set $T = T_0 \cup T_1$, vectors $\{\mathbf{d}_{i,j}\}_{(i,j)\in T}$, uncompressed ciphertexts $\{\gamma_{i,j}\}_{(i,j)\in T}$, and openings $\{\delta_{i,j}\}_{(i,j)\in T}$ and $\{\phi_{i,j}\}_{(i,j)\in T}$ from the sender.
- For all $(i,j) \in T$ do the following:
  - If $1 \neq \mathsf{VC.Verify}(\mathsf{crs}_{\mathsf{VC}}, \mathsf{com}, \gamma_{i,j}, (i,j), \delta_{i,j})$ or $1 \neq \mathsf{VC.Verify}(\mathsf{crs}_{\mathsf{VC}}, \mathsf{com}', \mathbf{d}_{i,j}, (i,j), \phi_{i,j})$ abort the protocol.
  - Decrypt $b'_{i,j} \leftarrow \mathsf{TDH.LocDec}(\mathsf{td}, \mathbf{d}_{i,j}, \gamma_{i,j})$.
  - If $(i,j) \in T_{b'_{i,j}}$ and $(i,j) \notin S$, add $(i,j) \in \bar{T}$.
  - Else if $(i,j) \in T_{1-b'_{i,j}}$ and $(i,j) \in S$, add $(i,j) \in \bar{T}$.
- Output $\bar{T}$ to the receiver.

Receiver's output. A seed $\mathsf{seed} \in \{0,1\}^\lambda$, hash values $\{h_i\}_{i\in[\nu]}$, a punctured key K^*, LOT ciphertexts $\{\mathsf{lotct}_{b,i,j}\}_{\beta\in\{0,1\},(i,j)\in S}$ and a set $\bar{T}$.

The following lemma states that the functionality described above can be implemented using a two-round protocol with communication sublinear in $\nu^2 = \mu$.

Lemma 6. *The functionality $\mathcal{G}$ can be implemented using a two-round NISC protocol with communication complexity of $\nu \cdot \mathsf{poly}(n, L, t_1, t_2, \lambda)$.*

Proof. By Lemma 1, it is enough to show that there is a circuit $\mathcal{C}$ with size $\nu \cdot \mathsf{poly}(n, L, t_1, t_2, \lambda)$ that implements the functionality described above.

To analyze the size of the circuit, we analyze each component individually. The total size of $\mathcal{C}$ will be the sum of all these components.

F_{Cons} can be implemented using a circuit of size $\nu \cdot L \cdot \mathsf{poly}(\lambda)$ (this is just the equality circuit).

For F_{CDS}, note that each $\mathsf{ek}_i \leftarrow \mathsf{TDH.KeyGen}(\mathsf{hk}, \mathbf{s}, \mathbf{t}_i)$ can be implemented using a circuit of size $n \cdot L \cdot \mathsf{poly}(\lambda)$ by the definition of $\mathsf{TDH.KeyGen}$, that is, independent of ν. Thus, repeating this process ν times, we conclude that F_{CDS} can be implemented using a circuit of size $\nu \cdot n \cdot L \cdot \mathsf{poly}(\lambda)$.

Analogously, for F_{CDS}, we have that each $h_i \leftarrow \mathsf{TDH.H}(\mathsf{hk}, \mathbf{r}_i)$ can be implemented using a circuit of size $L \cdot \mathsf{poly}(\lambda)$. Thus F_{CDS} can be implemented using a circuit of size $\nu \cdot L \cdot \mathsf{poly}(\lambda)$.

Let $t_1 = |S| = |\mathsf{Supp}(\mathbf{e})|$. To puncture the K on the set $\{(i,j,b) : (i,j) \in S, b \in \{0,1\}\}$ a circuit of size $\mathsf{poly}(t_1, \lambda)$ is needed. Moreover, for a fixed (i,j,b), $\mathsf{lotct}_{b,i,j} \leftarrow \mathsf{LOT.Enc}(\mathsf{crs}, h_{\mathsf{LOT}}, j, (\mathsf{K}, i, b))$ can be implemented using a circuit of size $\mathsf{poly}(L, \lambda)$ by the definition of LOT. Repeating this process t_1 times, we obtain a circuit of size $\mathsf{poly}(L, t_1, \lambda)$. Moreover, fetching each $\mathbf{r}_i$ requires $\nu \cdot \mathsf{poly}(\lambda)$. Hence the total size of this circuit is $\nu \cdot \mathsf{poly}(L, t_1, \lambda)$.

For F_{SendErr} :, note that each $b'_{i,j} \leftarrow \mathsf{TDH.LocDec}(\mathbf{s}, \mathbf{d}_{i,j}, \gamma_{i,j})$ requires a circuit of size $\mathsf{poly}(n, \lambda)$. Repeating this $|T|$ (where $|T| = |T_0 \cup T_1| \leq 2 \cdot t_2$) we obtain a circuit of size $\mathsf{poly}(n, t_2)$.

Finally, the total size of the circuit can be upperbounded by $\nu \cdot \mathsf{poly}(n, L, t_1, t_2, \lambda)$ and the result follows. □

5.2 Universally Composable Oblivious Transfer with Optimal Rate

We are now ready to present our OT scheme. We first present the scheme, then we analyze it.

Construction 2. *Let $L, n, \mu, t_1, t_2 \in \mathsf{poly}(\lambda)$. Let $\nu = \sqrt{\mu}$. Let $\chi_{\mu,t}$ be the uniform distribution over the binary vectors of size μ and hamming weight t. In the scheme, parties execute μ independent OTs. We now describe the scheme in full detail.*

$\mathsf{Setup}(1^\lambda)$:

- *Run* $\mathsf{hk} \leftarrow \mathsf{TDH.Setup}(1^\lambda, L)$.
- *Sample vectors* $\hat{\mathbf{t}}_1, \ldots, \hat{\mathbf{t}}_\nu \leftarrow\!\!\$\ \{0,1\}^\ell$ *and a matrix* $\mathbf{V} \leftarrow\!\!\$\ \mathbb{F}_2^{\ell \times L}$, *for* $i = 1, \ldots, \nu$ *set* $\mathbf{t}_i \leftarrow \hat{\mathbf{t}}_i \cdot \mathbf{V}$. *Additionally sample an LPN matrix* $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_\nu) \leftarrow\!\!\$\ \{0,1\}^{n \times \mu}$, *where each* $\mathbf{D}_i \in \{0,1\}^{n \times \nu}$.
- *Compute* $\mathsf{crs}_{\mathsf{LOT}} \leftarrow \mathsf{LOT.Setup}(1^\lambda)$.
- *Output* $\mathsf{crs} = (\mathsf{hk}, \{\mathbf{t}_i\}_{i \in [\nu]}, \mathbf{D}, \mathsf{crs}_{\mathsf{LOT}})$

$\mathsf{R}_1(\mathsf{crs}, \mathbf{b} \in \{0,1\}^\mu)$:

- *Parse* crs *as* $(\mathsf{hk}, \{\mathbf{t}_i\}_{i \in [\sqrt{\mu}]}, \mathbf{D}, \mathsf{crs}_{\mathsf{LOT}})$ *and* $\mathbf{b}$ *as* $(b_{1,1}, b_{1,2}, \ldots, b_{\nu,\nu})$.
- *Sample* $\mathbf{s} \leftarrow\!\!\$\ \{0,1\}^n$ *and* $\mathbf{e} \leftarrow\!\!\$\ \chi_{\mu,t}$.
- *Compute* $\mathbf{c} = \mathbf{s} \cdot \mathbf{D} + \mathbf{e} + \mathbf{b}$ *and* $(\mathsf{ek}, \mathsf{td}) \leftarrow \mathsf{TDH.KeyGen}(\mathsf{hk}, \mathbf{s}, (\mathbf{t}_1, \ldots, \mathbf{t}_\nu); r)$ *using random coins* $r \leftarrow\!\!\$\ \{0,1\}^\lambda$.
- *Compute* $h_{\mathsf{LOT}} \leftarrow \mathsf{LOT.H}(\mathsf{crs}_{\mathsf{LOT}}, (\mathbf{t}_1, \ldots, \mathbf{t}_\nu \cdot))$.
- *For all* $i \in [\nu]$ *compute* $\mathsf{ek}'_i \leftarrow \mathsf{TDH.Eval}(\mathsf{ek}, \mathbf{D}_i, \mathbf{c}_i)$ *and set* $\mathsf{ek}'_i = (\gamma_{i,1}, \ldots, \gamma_{i,\nu})$. *Compute* $\mathsf{com} \leftarrow \mathsf{VC.Com}(\mathsf{crs}_{\mathsf{VC}}, \mathsf{EK})$ *where* $\mathsf{EK} = (\mathsf{ek}_1, \ldots, \mathsf{ek}_\nu)$ *and* $\mathsf{com}' \leftarrow \mathsf{VC.Com}(\mathsf{crs}_{\mathsf{VC}}, \mathbf{D})$.
- *Set* $S = \mathsf{Supp}(\mathbf{e})$. *Send* $(S, \mathbf{s}, r, (\mathbf{t}_1, \ldots, \mathbf{t}_\nu), \mathsf{hk}, h_{\mathsf{LOT}}, \mathsf{com}, \mathsf{com}')$ *to* $\mathcal{G}_{\mathsf{aux}}$.
- *Output* $\mathsf{ot}_1 = (\mathsf{ek}, \mathbf{c})$ *and* $\mathsf{st} = \mathsf{td}$.

$\mathsf{S}(\mathsf{ot}_1, (\mathbf{m}_0, \mathbf{m}_1) \in \{0,1\}^\mu \times \{0,1\}^\mu)$:

- *Parse* $\mathbf{m}_0 = (m_{0,1,1}, \ldots, m_{0,\nu,\nu})$ *and* $\mathbf{m}_1 = (m_{1,1,1}, \ldots, m_{1,\nu,\nu})$. *Parse* ot_1 *as* $(\mathsf{ek}, \mathbf{c})$ *where* $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_\nu)$ *and* $\mathbf{c}_i \in \{0,1\}^\nu$.
- *For all* $i \in [\nu]$ *sample* $\mathbf{r}_i \leftarrow\!\!\$\ \{0,1\}^L$. *Compute* $h_i \leftarrow\!\!\$\ \mathsf{TDH.H}(\mathsf{hk}, \mathbf{r}_i)$. *Compute* $\mathsf{ek}'_i \leftarrow \mathsf{TDH.Eval}(\mathsf{ek}, \mathbf{D}_i, \mathbf{c}_i)$. *Finally compute* $\mathbf{z}_i \leftarrow \mathsf{TDH.Enc}(\mathsf{ek}'_i, \mathbf{r}_i)$.
- *Sample a puncturable PRF key* $\mathsf{K} \leftarrow\!\!\$\ \{0,1\}^\lambda$.
- *For all* $i \in [\nu]$ *parse* $\mathbf{z}_i = (z_{i,1}, \ldots, z_{i,\nu})$. *For all* $\beta \in \{0,1\}$ *sample* $\mathbf{f}_\beta \leftarrow\!\!\$\ \chi_{\mu,t_2}$ *such that* $\mathbf{f} = (f_{i,j})_{i,j \in [\nu]}$. *For all* $j \in [\nu]$ *compute*
$$w_{0,i,j} = z_{i,j} + m_{0,i,j} + f_{0,i,j} + \mathsf{PRF.Eval}(\mathsf{K}, (i,j,0)) \bmod 2$$
and
$$w_{1,i,j} = z_{i,j} + (\mathbf{r}_i \cdot \mathbf{t}_j^T) + m_{1,i,j} + f_{1,i,j} + \mathsf{PRF.Eval}(\mathsf{K}, (i,j,1)) \bmod 2.$$
- *Sample a PRG seed* $\mathsf{seed} \leftarrow\!\!\$\ \{0,1\}^\lambda$ *and compute* $\bar{\mathbf{w}} = (w_{0,1,1}, w_{1,1,1}, w_{0,1,2}, \ldots, w_{1,\nu,\nu}) + \mathsf{PRG}(\mathsf{seed})$.
- *Compute* $h_{\mathsf{LOT}} \leftarrow \mathsf{LOT.H}(\mathsf{crs}_{\mathsf{LOT}}, (\mathbf{t}_1, \ldots, \mathbf{t}_\nu))$.
- *Set* $T_0 = \mathsf{Supp}(\mathbf{f}_0)$, $T_1 = \mathsf{Supp}(\mathbf{f}_1)$ *and* $T = T_0 \cup T_1$.
- *For all* $i \in [\nu]$, *set* $\mathsf{ek}'_i = (\gamma_{i,1}, \ldots, \gamma_{i,\nu})$. *Compute* $(\mathsf{com}, \mathsf{st}_{\mathsf{VC}}) \leftarrow \mathsf{VC.Com}(\mathsf{crs}_{\mathsf{VC}}, \mathsf{EK})$ *where* $\mathsf{EK} = (\mathsf{ek}_1, \ldots, \mathsf{ek}_\nu)$. *Additionally, for all* $(i,j) \in T$ *compute* $\delta_{i,j} \leftarrow \mathsf{VC.Open}(\mathsf{crs}, \mathsf{com}, \mathsf{st}, (i,j))$. *Additionally, compute* $(\mathsf{com}', \mathsf{st}'_{\mathsf{VC}}) \leftarrow \mathsf{VC.Com}(\mathsf{crs}_{\mathsf{VC}}, \mathbf{D})$ *and for all* $(i,j) \in T$ *compute* $\phi_{i,j} \leftarrow \mathsf{VC.Open}(\mathsf{crs}, \mathsf{com}', \mathsf{st}', (i,j))$.
- *Send* $(\mathsf{ek}, \mathsf{seed}, \{h_i \mathbf{r}_i, \mathbf{t}_i\}_{i, \in [\nu]}, \mathsf{hk}, T_0, T_1, \{\mathbf{d}_{i,j}, \gamma_{i,j}, \delta_{i,j}, \phi_{i,j}\}_{(i,j) \in T}, h_{\mathsf{LOT}})$ *to* $\mathcal{G}_{\mathsf{aux}}$.
- *Output* $\mathsf{ot}_2 = (\{h_i\}_{i \in [\nu]}, \bar{\mathbf{w}})$.

$\mathsf{R}_2(\mathsf{ot}_2, \mathsf{st})$:

- *Parse* ot_2 *as* $(\{h_i\}_{i\in[\nu]}, \bar{\mathbf{w}})$ *and* st *as* $(\{\mathsf{st}_i\}_{i\in[\mu]}$.
- *Obtain* $(\mathsf{seed}, \{h'_i\}_{i\in[\nu]}, \mathsf{K}^*, \{\mathsf{lotct}_{b,i,j}\}_{\beta\in\{0,1\},(i,j)\in S}, \bar{T})$ *from* $\mathcal{G}_{\mathsf{aux}}$.
- *Compute* $(w_{0,1,1}, w_{1,1,1}, w_{0,1,2}, \dots, w_{1,\nu,\nu}) = \bar{\mathbf{w}} + \mathsf{PRG}(\mathsf{seed})$.
- *If there is* $i \in [\nu]$ *such that* $h_i \neq h'_i$*, abort the protocol.*
- *For all* $i \in [\mu]$ *compute* $\mathbf{a}_i \leftarrow \mathsf{TDH.Dec}(\mathsf{td}, (\mathbf{D}_i, \mathbf{c}_i,), h_i)$*. Parse* $\mathbf{a}_i = (a_{i,1}, \dots, a_{i,\nu})$.
- *For all* $(i,j) \in S = \mathsf{Supp}(\mathbf{e})$ *and all* $\beta \in \{0,1\}$ *compute* $y_{i,j,b} \leftarrow \mathsf{LOT.Dec}(\mathsf{crs}_{\mathsf{LOT}}, \{\mathbf{t}_1, \dots, \mathbf{t}_\nu\}, \mathsf{lotct}_{\beta,i,j}, i)$.
- *For all* $i,j \in [\mu]$ *compute*

$$m'_{i,j} = \begin{cases} w_{b_{i,j},i,j} + a_{i,j} + y_{i,j,b_{i,j}} \bmod 2, \textit{ if } j \in \mathsf{Supp}(\mathbf{e}_i) \\ w_{b_{i,j},i,j} + a_{i,j} + \mathsf{PRF.Eval}(\mathsf{K}^*, (i,j,b_{i,j})) \bmod 2, \textit{ otherwise} \end{cases}.$$

- *Finally, output* $m_{i,j}$ *where*

$$m_{i,j} = \begin{cases} m'_{i,j} + 1 \bmod 2, \textit{ if } (i,j) \in \bar{T} \\ m'_{i,j}, \textit{ otherwise} \end{cases}.$$

Communication Complexity. We now analyze the communication complexity for our protocol. For this analysis, we instantiate the ideal functionality $\mathcal{G}_{\mathsf{aux}}$ using a two-round protocol as in Lemma 6. Recall that, using Lemma 6, the ideal functionality $\mathcal{G}_{\mathsf{aux}}$ can be instantiated using a two-round protocol with total communication complexity $\nu \cdot \mathsf{poly}(n, L, t_1, t_2, \lambda)$. Let $\mathsf{nisc}_1, \mathsf{nisc}_2$ be the receiver's and sender's message in this protocol, respectively.

- **Receiver's message.** The receiver's message is composed by $(\mathsf{ek}, \mathbf{c})$ and nisc_1 where
  - $|\mathsf{ek}| = \nu \cdot L \cdot n \cdot P_1(\lambda)$ for some polynomial P_1
  - $|\mathbf{c}| = \nu^2 = \mu$
  - $|\mathsf{nisc}_1| = \nu \cdot P_2(n, L, t_1, t_2, \lambda)$ for some polynomial P_2.

  Hence, the upload rate ρ_{up} of the protocol is

$$\rho_{\mathsf{up}} = \frac{\nu \cdot L \cdot n \cdot P_1(\lambda) + \nu^2 + \nu \cdot P_2(n, L, t_1, t_2, \lambda)}{\nu^2} = 1 + \frac{L \cdot n \cdot P_1(\lambda) + P_2(n, L, t_1, t_2, \lambda)}{\nu}.$$

  Setting ν, n, L, t_1, t_2 such that $o(\nu) = L \cdot n \cdot P_1(\lambda)$ and $o(\nu) = P_2(n, L, t_1, t_2)$[6] then $\rho_{\mathsf{up}} \to 1$ for large enough ν.

- **Sender's message.** The sender's message is composed by $(\{h_i\}_{i\in[\nu]}, \bar{\mathbf{w}})$ and nisc_2 where
  - $|\{h_i\}_{i\in[\nu]}| = \nu \cdot \mathsf{poly}(\lambda) = \nu \cdot n \cdot Q_1(\lambda)$ for some polynomial Q_1.
  - $|\bar{\mathbf{w}}| = 2 \cdot \nu^2 = 2 \cdot \mu$.
  - $|\mathsf{nisc}_2| = \nu \cdot \mathsf{poly}(n, L, t_1, t_2, \lambda) = \nu \cdot Q_2(n, L, t_1, t_2, \lambda)$ for some polynomial Q_2.

  Hence the download rate ρ_{down} of the protocol is

$$\rho_{\mathsf{down}} = \frac{\nu \cdot Q_1(\lambda) + 2\nu^2 + \nu \cdot Q_2(n, L, t_1, t_2, \lambda)}{\nu^2} = 2 + \frac{Q_1(\lambda) + Q_2(n, L, t_1, t_2, \lambda)}{\nu}.$$

  A similar choice of parameters as in the previous case, yields that $\rho_{\mathsf{up}} \to 2$ for large enough ν.

[6]For this to happen, we just have to set $o(\nu) = n, L, t_1, t_2$ accordingly.

Analysis. We now analyze the correctness and security of our scheme.

Theorem 2 (Correctness)**.** *The scheme presented in Construction 2 is correct.*

Proof. First, the receiver gets the seed seed and can recover $(w_{0,1,1}, w_{1,1,1}, w_{0,1,2}, \dots, w_{1,\mu,\mu}) = \bar{\mathbf{c}} + \mathsf{PRG}(\mathsf{seed})$.

By the correctness of the LOT, we have that $y_{i,j,\beta} = \mathsf{PRF.Eval}(\mathsf{K}, (i,j,\beta)) + (\mathbf{r}_i \cdot \mathbf{t}_j^T)$ for $(i,j) \in \mathsf{Supp}(\mathbf{e})$ and $\beta \in \{0,1\}$, where $y_{i,j,\beta} \leftarrow \mathsf{LOT.Dec}(\mathsf{crs}_{\mathsf{LOT}}, \{\mathbf{t}_1, \dots, \mathbf{t}_\nu\}, \mathsf{lotct}_{\beta,i,j}, i)$.

Moreover, the receiver obtains K^* (which is punctured at (i,j,β) for $(i,j) \in \mathsf{Supp}(\mathbf{e})$ and $\beta \in \{0,1\}$) and a set

$$\bar{T} = \left\{ (i,j) \in [\nu]^2 : \begin{array}{c} (i,j) \in T_{b'_{i,j}} \wedge (i,j) \notin S \\ \vee \\ (i,j) \in T_{1-b'_{i,j}} \wedge (i,j) \in S(i,j) \end{array} \right\}$$

where $b'_{i,j} \leftarrow \mathsf{TDH.LocDec}(\mathbf{s}, \mathbf{d}_{i,j}, \gamma_{i,j})$. Note that by the local correctness of the TDH, we have that $b'_{i,j} = b_{i,j} + e_{i,j}$.

Now recall that, by the correctness of TDH, we have that $a_{i,j} + z_{i,j} = b_{i,j} \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T)$ for all $i,j \in [\nu]$.

We divide the proof in several cases. In this first case, $(i,j) \notin \mathsf{Supp}(\mathbf{e})$. In this case we have that

$$w_{b_{i,j},i,j} = z_{i,j} + b_{i,j} \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T) + m_{b_{i,j},i,j} + f_{b_{i,j},i,j} + \mathsf{PRF.Eval}(\mathsf{K}, (i,j,b_{i,j})).$$

Thus

$$m'_{b_{i,j},i,j} = w_{b_{i,j},i,j} + a_{i,j} + \mathsf{PRF.Eval}(\mathsf{K}^*, (i,j,b_{i,j}) = m_{b_{i,j},i,j} + f_{b_{i,j},i,j}$$

where the equality holds from the correctness of TDH and the puncturable PRF.

If $f_{b_{i,j},i,j} = 1$ then $(i,j) \in \bar{T}$ and thus the receiver outputs $m_{b_{i,j},i,j} = m'_{b_{i,j},i,j} + 1 = m_{b_{i,j},i,j}$. Otherwise $m_{b_{i,j},i,j} = m'_{b_{i,j},i,j} = m_{b_{i,j},i,j}$.

Now we consider the case where $(i,j) \in \mathsf{Supp}(\mathbf{e})$. Recall that $y_{i,j,b_{i,j}} = \mathsf{PRF.Eval}(\mathsf{K}, (i,j,b_{i,j})) + (\mathbf{r}_i \cdot \mathbf{t}_j^T)$. This in this case, the receiver computes

$$\begin{aligned} w_{b_{i,j},i,j} + a_{i,j} + y_{i,j,b_{i,j}} &= \underbrace{(z_{i,j} + a_{i,j})}_{=(1-b_{i,j})(\mathbf{r}_i \cdot \mathbf{t}_j^T)} + b_{i,j}(\mathbf{r}_i \cdot \mathbf{t}_j^T) + m_{b_{i,j},i,j} + f_{i,j} + (\mathbf{r}_i \cdot \mathbf{t}_j^T) \\ &= m_{b_{i,j},i,j} + f_{i,j} \end{aligned}$$

In this case, if $f_{b_{i,j},i,j} = 1$ then $(i,j) \in \bar{T}$. To see this note that, since $b'_{i,j} = 1 - b_{i,j}$, $(i,j) \in T_{1-b'_{i,j}} = T_{b_{i,j}}$ and $(i,j) \in S$. The remaining of the analysis follows the same reasoning as in the previous case. $\square$

Theorem 3 (Security)**.** *The scheme presented in Construction 2 implements the functionality $\mathcal{F}_{\mathsf{OT}}$ in the $\mathcal{G}_{\mathsf{aux}}$ hybrid model.*

The proof of this theorem follows from Lemmas 7 and 8 presented below.

Lemma 7 (Receiver security)**.** *Assume that* TDH *is receiver secure,* VC *is position binding and that the LPN assumption holds. Then the scheme presented in Construction 2 is secure against malicious senders in the $\mathcal{G}_{\mathsf{aux}}$ hybrid model.*

Proof. We start the proof by presenting the simulator $\mathsf{Sim}_\mathsf{S}(1^\lambda)$ for a corrupted sender.

$\mathsf{Sim}_\mathsf{S}(\lambda)$:

- Simulate the functionality $\mathcal{G}_{\mathsf{aux}}$, and create the crs as in the real scheme.
- For all $i \in [\mu]$ compute $(\mathsf{ek}, \mathsf{st}) \leftarrow \mathsf{TDHKeyGen}(\mathsf{hk}, \mathbf{0}; r_i)$ and sample $\mathbf{c} \leftarrow\!\!\$\ \{0,1\}^\mu$.
- Send $\mathsf{ot}_1 = \{\mathsf{CT}_i, \mathbf{c}_i\}_{i \in [\mu]}$ to the sender.

- Upon receiving a query $(\mathsf{ek}, \mathsf{seed}, \{h_i, \mathbf{r}_i, \mathbf{t}'_i\}_{i,\in[\nu]}, \mathsf{hk}', T_0, T_1, \{\mathbf{d}_{i,j}, \gamma_{i,j}, \delta_{i,j}\}_{(i,j)\in T}, h_{\mathsf{LOT}})$ (intended to $\mathcal{G}_{\mathsf{aux}}$ and a message $(\{h_i\}_{i\in[\nu]}, \bar{\mathbf{w}})$ from the sender do the following:
  1. If $\mathsf{hk}' \neq \mathsf{hk}$ or $(\mathbf{t}_1, \dots, \mathbf{t}_\nu) \neq (\mathbf{t}'_1, \dots, \mathbf{t}'_\nu)$ or $h_{\mathsf{LOT}} \neq \mathsf{LOT.H}(\mathsf{crs}, \{\mathbf{t}'_1, \dots, \mathbf{t}'_\nu\})$ abort the protocol.
  2. If there is $i \in [\nu]$ such that $h_i \neq \mathsf{TDH.H}(\mathsf{hk}, \mathbf{r}_i)$ abort the protocol.
  3. Compute $(w_{0,1,1}, w_{1,1,1}, w_{0,1,2}, \dots, w_{1,\mu,\mu}) = \bar{\mathbf{w}} + \mathsf{PRG}(\mathsf{seed})$.
  4. For all $i \in [\nu]$ compute $\mathsf{ek}'_i \leftarrow \mathsf{TDH.Eval}(\mathsf{ek}, \mathbf{D}_i, \mathbf{c}_i)$, and $\mathbf{z}_i \leftarrow \mathsf{TDH.Enc}(\mathsf{ek}'_i, \mathbf{r}_i)$. Parse $\mathsf{ek}'_i = (\gamma'_{i,1}, \dots, \gamma'_{i,1})$. For $(i,j) \in T$, if $\gamma_{i,j} \neq \gamma'_{i,j}$ abort the protocol.
  5. For all $\beta \in \{0,1\}$ let $\mathbf{f}_\beta$ be the vector such that $\mathsf{Supp}(\mathbf{f}_\beta) = T_\beta$.
  6. For all $i, j \in [\nu]$ set
$$m_{0,i,j} = w_{0,i,j} + z_{i,j} + f_{0,i,j} + \mathsf{PRF.Eval}(\mathsf{K}, (i,j,0))$$
and
$$m_{1,i,j} = w_{1,i,j} + z_{i,j} + (\mathbf{r}_i \cdot \mathbf{t}_j^T) + f_{1,i,j} + \mathsf{PRF.Eval}(\mathsf{K}, (i,j,1)).$$
- It sends $(\mathbf{m}_0, \mathbf{m}_1)$ to $\mathcal{F}_{\mathsf{OT}}$ where $\mathbf{m}_0 = (m_{0,i,j})_{i,j\in[\nu]}$ and $\mathbf{m}_1 = (m_{1,i,j})_{i,j\in[\nu]}$.

We now prove that the real-world and ideal-world execution are indistinguishable. The proof follows from the following sequence of hybrids.

Hybrid $\mathcal{H}_0$. This is the real-world execution.

Hybrid $\mathcal{H}_1$. In this hybrid, the simulator checks if $\gamma_{i,j} \neq \gamma'_{i,j}$ where $\mathsf{ek}'_i = (\gamma'_{i,1}, \dots, \gamma'_{i,1})$, for $(i,j) \in T$, instead of running VC.Verify. Additionally, check if $\mathbf{d}'_{i,j} = \mathbf{d}_{i,j}$.

Indistinguishability of hybrids follows from the position binding of the underlying VC.

Hybrid $\mathcal{H}_2$. In this hybrid, the simulator extracts $\mathbf{m}'_0, \mathbf{m}'_1$ as described in the simulation. If there exists an index $(i,j) \in [\nu] \times [\nu]$ such that the message $m_{b_{i,j}}$ output by the honest receiver is different from the extracted $m'_{b_{i,j}}$, then the simulator aborts the protocol.

To see that these hybrids are indistinguishable, we will show that the output distribution is the same in both executions. Assume that in the real-world, the receiver's choice bit for a position $(i,j) \in [\nu] \times [\nu]$ is $b_{i,j}$. We analyze four different possible cases.

First, assume that $(i,j) \notin S$ and $(i,j) \notin T$. In this case the real-world receiver extracts
$$\begin{aligned} m_{b_{i,j},i,j} &= w_{b_{i,j},i,j} + a_{i,j} + \mathsf{PRF.Eval}(\mathsf{K}^*, (i,j,b_{i,j})) \\ &= w_{b_{i,j},i,j} + z_{i,j} + b_{i,j} \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T) + \mathsf{PRF.Eval}(\mathsf{K}, (i,j,b_{i,j})) \end{aligned}$$
by the correctness of the TDH and the PRF. Thus, the receiver obtains the same messages that the sender extracts.

In the second case $(i,j) \in S$ and $(i,j) \notin T$. In this case
$$\begin{aligned} m_{b_{i,j},i,j} &= w_{b_{i,j},i,j} + a_{i,j} + y_{i,j,b_{i,j}} \\ &= w_{b_{i,j},i,j} + z_{i,j} + (b_{i,j}+1) \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T) + \mathsf{PRF.Eval}(\mathsf{K}, (i,j,b_{i,j})) + (\mathbf{r}_i \cdot \mathbf{t}_j^T) \\ &= w_{b_{i,j},i,j} + z_{i,j} + b_{i,j} \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T) + \mathsf{PRF.Eval}(\mathsf{K}, (i,j,b_{i,j})) \end{aligned}$$
where the second equality holds by the correctness of the LOT.

In the third and last case $(i,j) \in T_{b_{i,j}}$, meaning that $f_{b_{i,j},i,j} = 1$. If $(i,j) \notin S$, $\mathsf{TDH.LocDec}$ outputs $b_{i,j}$ and $(i,j) \in \bar{T}$. Hence the real-world receiver outputs
$$\begin{aligned} m_{b_{i,j},i,j} &= w_{b_{i,j},i,j} + a_{i,j} + \mathsf{PRF.Eval}(\mathsf{K}^*, (i,j,b_{i,j})) + 1 \\ &= w_{b_{i,j},i,j} + z_{i,j} + b_{i,j} \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T) + \mathsf{PRF.Eval}(\mathsf{K}, (i,j,b_{i,j})) + f_{b_{i,j},i,j}. \end{aligned}$$

Analogously, if $(i,j) \notin S$, $\mathsf{TDH.LocDec}$ outputs $1-b_{i,j}$ and $(i,j) \in \bar{T}$ (by definition of $\bar{T}$, (i,j) is added if $(i,j) \in T_{1-(1-b_{i,j})}$. Hence,

$$\begin{aligned} m_{b_{i,j},i,j} &= w_{b_{i,j},i,j} + a_{i,j} + y_{i,j,b_{i,j}} + 1 \\ &= w_{b_{i,j},i,j} + z_{i,j} + (b_{i,j}+1)\cdot(\mathbf{r}_i \cdot \mathbf{t}_j^T) + \mathsf{PRF.Eval}(\mathsf{K},(i,j,b_{i,j})) + (\mathbf{r}_i \cdot \mathbf{t}_j^T) + f_{b_{i,j},i,j} \\ &= w_{b_{i,j},i,j} + z_{i,j} + b_{i,j}\cdot(\mathbf{r}_i \cdot \mathbf{t}_j^T) + \mathsf{PRF.Eval}(\mathsf{K},(i,j,b_{i,j})) + f_{b_{i,j},i,j}. \end{aligned}$$

Hybrid $\mathcal{H}_3$. In this hybrid, the simulator computes $(\mathsf{ek},\mathsf{st}) \leftarrow \mathsf{TDHKeyGen}(\mathsf{hk},\mathbf{0};r)$. Indistinguishability follows from the receiver security of the underlying TDH.

Hybrid $\mathcal{H}_4$. In this hybrid, we replace $\mathbf{c}_i$ by $\mathbf{c}_i \leftarrow\$ \{0,1\}^\lambda$. Indistinguishability follows from the LPN assumption.

This concludes the proof as the outputs are statistically indistinguishable. □

Lemma 8 (Sender security). *Assume that* TDH *is sender secure,* PRF *is pseudorandom at punctured points,* LOT *is sender secure and that the (entropic) LPN assumption holds. Then the scheme presented in Construction 2 is secure against malicious receivers in the* $\mathcal{G}_{\mathsf{aux}}$ *hybrid model.*

Proof. We start the proof by presenting the simulator $\mathsf{Sim}_\mathsf{R}(1^\lambda)$ for a corrupted receiver.

$\mathsf{Sim}_\mathsf{R}(\lambda)$:

- Simulate the ideal functionality $\mathcal{G}_{\mathsf{aux}}$. Create crs as in the real scheme.
- For all $i \in [\nu]$ sample $\mathbf{r}_i \leftarrow\$ \{0,1\}^L$ and compute $h_i \leftarrow \mathsf{TDH.H}(\mathsf{hk},\mathbf{r}_i)$.
- Upon receiving a query $(S,\mathbf{s},r,(\mathbf{t}'_1,\dots,\mathbf{t}'_\nu),\mathsf{hk},h_{\mathsf{LOT}})$ (intended to $\mathcal{G}_{\mathsf{aux}}$) and a message $(\mathsf{ek},\mathbf{c})$ from the receiver do the following:
  1. If $h_{\mathsf{LOT}} \neq \mathsf{LOT.H}(\mathsf{crs},\{\mathbf{t}_1,\dots,\mathbf{t}_\nu\})$ or $(\mathbf{t}'_1,\dots,\mathbf{t}'_\nu) \neq (\mathbf{t}_1,\dots,\mathbf{t}_\nu)$ abort the protocol.
  2. If $\mathsf{ek} \neq \mathsf{TDH.KeyGen}(\mathsf{hk},\mathbf{s};r)$, return $(\{h_i\}_{i\in[\nu]}, \bar{\mathbf{w}} \leftarrow\$ \{0,1\}^{2\mu^2})$ to the receiver. Else continue.
  3. Let $\mathbf{e} \in \{0,1\}^\mu$ such that $\mathsf{Supp}(\mathbf{e}) = S$. Compute
  $$\mathbf{b} = \mathbf{c} + \mathbf{s}\cdot\mathbf{D} + \mathbf{e} \bmod 2.$$
- Send $\mathbf{b} = (\mathbf{b}_1,\dots,\mathbf{b}_\mu)$ to $\mathcal{F}_{\mathsf{OT}}$.
- Upon receiving $\mathbf{m} = \mathbf{m}_\mathbf{b}$ do the following:
  1. For all $i \in [\mu]$, compute $\mathbf{a}_i \leftarrow \mathsf{TDH.Dec}(\mathsf{st},(\mathbf{D},\mathbf{c}_i),\{\mathbf{t}_j\}_{j\in[\mu]},h_i)$.
  2. Sample $\mathbf{f}_0,\mathbf{f}_1 \leftarrow\$ \chi_{\mu,t_2}$.
  3. Sample a PRF key $\mathsf{K} \leftarrow \mathsf{PRF'KeyGen}(1^\lambda)$ and compute $\mathsf{K}^* \leftarrow \mathsf{PRF.Puncture}(\mathsf{K},\{(i,j,\beta\}_{(i,j)\in S,\beta\in\{0,1\}})$.
  4. For all $i,j \in [\nu]$ set
  $$w_{b_{i,j},i,j} = \begin{cases} a_{i,j} + m_{b_{i,j},i,j} + \mathsf{PRF.Eval}(\mathsf{K},(i,j,b_{i,j})) + f_{b_{i,j},i,j}, \text{ if } (i,j)\notin S \\ a_{i,j} + m_{b_{i,j},i,j} + u_{b_{i,j},i,j} + f_{b_{i,j},i,j}, \text{ otherwise} \end{cases}$$
  where $u_{b_{i,j},i,j} \leftarrow\$ \{0,1\}$. Additionally, set $w_{1-b_{i,j},i,j} \leftarrow\$ \{0,1\}$.
  5. For $(i,j) \in S$ compute $\mathsf{lotct}_{i,j,\beta} \leftarrow \mathsf{LOTSim}(\mathsf{crs},h_{\mathsf{LOT}},j,u_{\beta,i,j})$ where $u_{1-b_{i,j},i,j} \leftarrow\$ \{0,1\}$.
  6. Set $\bar{T} = \{(i,j) : f_{b_{i,j},i,j} = 1\}$.
  7. Sample a PRG seed $\mathsf{seed} \leftarrow\$ \{0,1\}^\lambda$ and set $\bar{\mathbf{c}} = (w_{0,1,1},w_{1,1,1},w_{0,1,2},\dots,w_{1,\mu,\mu}) + \mathsf{PRG}(\mathsf{seed})$.

8. Output $(\mathsf{seed}, \{h'_i\}_{i\in[\nu]}, \mathsf{K}^*, \{\mathsf{lotct}_{\beta,i,j}\}_{\beta\in\{0,1\},(i,j)\in S}, \bar{T})$ via $\mathcal{G}_{\mathsf{aux}}$ to the receiver.

- Output $\mathsf{ot}_2 = (\{h_i\}_{i\in[\mu]}, \bar{\mathbf{c}})$.

We now show that the real-world and ideal-world executions are insdistinguishable. This follows from the following sequence of hybrids.

Hybrid $\mathcal{H}_0$. This hybrid is the real-world execution.

Hybrid $\mathcal{H}_1$. In this hybrid, the simulator simulates $\mathcal{G}_{\mathsf{aux}}$ as in the real experiment, except that if $\mathsf{ek} \neq \mathsf{TDH.KeyGen}(\mathsf{hk}, \mathbf{s}; r)$, it sets $\bar{\mathbf{c}} \leftarrow\$ \{0,1\}^{2\mu}$.

Indistinguishability of hybrids follows from the security of the PRG.

Hybrid $\mathcal{H}_2$. In this hybrid, the simulator extracts $\mathbf{b}$ as described in the simulation and computes

$$w_{\beta,i,j} = a_{i,j} + (\beta + b_{i,j} + e_{i,j}) \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T) + f_{\beta,i,j} + \mathsf{PRF.Eval}(\mathsf{K}, (i,j,\beta)) + m_{\beta,i,j} \mod 2$$

for all $i,j \in [\nu]$ and $\beta \in \{0,1\}$.

The hybrids are identical since by the correctness of the TDH we have that $z_{i,j} = a_{i,j} + (\mathbf{s}\cdot\mathbf{d}_{i,j}^T + c_{i,j}) \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T) = a_{i,j} + (b_{i,j} + e_{i,j}) \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T)$.

Hybrid $\mathcal{H}_3$. In this hybrid, the simulator computes $y_{\beta,i,j} \leftarrow \mathsf{PRF}(\mathsf{K}, (i,j,\beta_{i,j})) + (\mathbf{r}_i \cdot \mathbf{t}_j^T)$ and $\mathsf{lotct}_{i,j,\beta} \leftarrow \mathsf{LOT.Sim}(\mathsf{crs}, h_{\mathsf{LOT}}, j, y_{\beta,i,j})$ for all $\beta \in \{0,1\}$ and all $(i,j) \in S$.

Indistinguishability of hybrids folllows from the sender security of the underlying LOT scheme.

Hybrid $\mathcal{H}_4$. In this hybrid, the simulator sets $y_{\beta,i,j} = u_{\beta,i,j} + (\mathbf{r}_i \cdot \mathbf{t}_j^T)$ where $u_{\beta,i,j} \leftarrow\$ \{0,1\}$ for $i,j \in S$ and $\beta \in \{0,1\}$ and sets

$$w_{\beta,i,j} = a_{i,j} + (\beta + b_{i,j} + 1) \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T) + f_{\beta,i,j} + u_{\beta,i,j} + m_{\beta,i,j}.$$

Indistinguishability of hybrids follows from the pseudorandomness of the PRF at punctured points.

Hybrid $\mathcal{H}_5$. In this hybrid, the simulator computes

$$w_{\beta,i,j} = a_{i,j} + (\beta + b_{i,j}) \cdot (\mathbf{r}_i \cdot \mathbf{t}_j^T) + f_{\beta,i,j} + u_{\beta,i,j} + m_{\beta,i,j}$$

where $u_{\beta,i,j} \leftarrow\$ \{0,1\}$ and sets $y_{\beta,i,j} = u_{\beta,i,j}$ for $i,j \in S$ and $\beta \in \{0,1\}$.

The hybrids are identical.

Hybrid $\mathcal{H}_6$. In this hybrid, the simulator chooses $\hat{\mathbf{r}}_1, \ldots, \hat{\mathbf{r}}_\nu \leftarrow\$ \{0,1\}^\ell$ uniformly at random and sets

$$w_{\beta,i,j} = a_{i,j} + (\beta + b_{i,j}) \cdot (\hat{\mathbf{r}}_i \cdot \hat{\mathbf{t}}_j^T) + f_{\beta,i,j} + u_{\beta,i,j} + m_{\beta,i,j}$$

for all $i,j \in [\mu]$ and $\beta \neq b_{i,j}$.

Indistinguishability of hybrids $\mathcal{H}_5$ and $\mathcal{H}_6$ follows via the leftover hash lemma. Note that it holds that $\mathbf{t}_i = \hat{\mathbf{t}}_i \cdot \mathbf{V}$. Hence it holds that

$$\mathbf{r}_i \cdot \mathbf{t}_j^\top = (\mathbf{r}_i \cdot \mathbf{V}^\top)\hat{\mathbf{t}}_j^\top.$$

Now, given $h_i = \mathsf{TDH.H}(\mathsf{hk}, \mathbf{r}_i) \in \{0,1\}^\lambda$, it holds by the min-entropy chain rule that the conditional min-entropy of $\mathbf{r}_i$ given h_i is at least $L - \lambda$ bits. As the matrix $\mathbf{V} \in \mathbb{F}_2^{L\times\ell}$ is chosen uniformly random and importantly *independently of* hk, $\mathbf{r}_i$ and therefore h_i, it holds by the leftover hash Lemma that

$$(h_i, \mathbf{r}_i \cdot \mathbf{V}^\top) \approx_s (h_i, \hat{\mathbf{r}}_i),$$

for a uniformly random $\hat{\mathbf{r}}_i \in \{0,1\}^\ell$. Hence the claim follows.

Hybrid $\mathcal{H}_7$. In this hybrid the simulator sets

$$w_{\beta,i,j} = \mathbf{u}_{\beta,i,j}$$

for $\beta \neq b_{i,j}$ for all $i, j \in [\mu]$.

Indistinguishability of hybrids follows from the hardness of LPN. To see this note that in hybrid $\mathcal{H}_6$ for all $(i,j) \in [\nu] \times [\nu]$ and $\beta \neq b_{i,j}$ we have that

$$w_{\beta,i,j} = a_{i,j} + (\hat{\mathbf{r}}_i \cdot \hat{\mathbf{t}}_j^T) + f_{\beta,i,j} + u_{\beta,i,j} + m_{\beta,i,j}.$$

Since the $\hat{\mathbf{r}}_i$ are uniformly random and independent of all other protocol messages, we can argue by the LPN assumption that

$$(\hat{\mathbf{t}}_1, \dots, \hat{\mathbf{t}}_\nu, (\hat{\mathbf{r}}_i \cdot \hat{\mathbf{t}}_j^T + f_{\beta,i,j})_{i,j}) \approx_c (\hat{\mathbf{t}}_1, \dots, \hat{\mathbf{t}}_\nu, (\mathbf{u}'_{\beta,i,j})_{i,j})$$

for uniformly random $\mathbf{u}'_{\beta,i,j}$. It follows that

$$w_{\beta,i,j} = a_{i,j} + \mathbf{u}'_{\beta,i,j} + u_{\beta,i,j} + m_{\beta,i,j}$$

is uniformly random.

Hybrid $\mathcal{H}_8$. The simulator computes $\bar{T}$ as $\bar{T} = \{(i,j) : f_{b_{i,j},i,j} = 1\}$. The hybrids are statistically indistinguishable by the definition of $\bar{T}$.

Note that the last hybrid corresponds to the simulation described above. This concludes the proof. $\square$

Acknowledgements

Pedro Branco is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project number 537717419 and partially funded by the German Federal Ministry of Education and Research (BMBF) in the course of the 6GEM research hub under grant number 16KISK038. Nico Döttling: Funded by the European Union (ERC, LACONIC, 101041207). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[ADD+22] Divesh Aggarwal, Nico Döttling, Jesko Dujmovic, Mohammad Hajiabadi, Giulio Malavolta, and Maciej Obremski. Algebraic Restriction Codes and Their Applications. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

[BBD+20] Zvika Brakerski, Pedro Branco, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Constant ciphertext-rate non-committing encryption from standard assumptions. In *TCC 2020: 18th Theory of Cryptography Conference, Part I*, Lecture Notes in Computer Science, pages 58–87. Springer, Heidelberg, Germany, March 2020.

[BBDP22] Zvika Brakerski, Pedro Branco, Nico Döttling, and Sihang Pu. Batch-OT with optimal rate. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 157–186, Cham, 2022. Springer International Publishing.

[BCG$^+$19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 291–308. ACM Press, November 11–15, 2019.

[BCG$^+$19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 489–518, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[BCG$^+$20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st Annual Symposium on Foundations of Computer Science*, pages 1069–1080. IEEE Computer Society Press, 2020.

[BCG$^+$20b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, Lecture Notes in Computer Science, pages 387–416, Santa Barbara, CA, USA, August 16–20, 2020. Springer, Heidelberg, Germany.

[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.

[BDGM19] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 407–437, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany.

[BDS23] Pedro Branco, Nico Döttling, and Akshayaram Srinivasan. A framework for statistically sender private OT with optimal rate. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 548–576, Cham, 2023. Springer Nature Switzerland.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 479–488. ACM, 1996.

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 1–20, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. 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.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300, Bengalore, India, December 1–5, 2013. 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.

[CDG$^+$17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 33–65, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 55–72, Nara, Japan, February 26 – March 1, 2013. Springer, Heidelberg, Germany.

[CGH$^+$21] Melissa Chase, Sanjam Garg, Mohammad Hajiabadi, Jialin Li, and Peihan Miao. Amortizing rate-1 OT and applications to PIR and PSI. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography*, pages 126–156, Cham, 2021. Springer International Publishing.

[DG17] Nico Döttling and Sanjam Garg. From selective IBE to full IBE and selective HIBE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 372–408, Baltimore, MD, USA, November 12–15, 2017. 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 Vincent Rijmen and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, Lecture Notes in Computer Science, pages 768–797. Springer, Heidelberg, Germany, May 2020.

[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.

[DGI$^+$19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 3–32, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 93–122, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986.

[GH19] Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 438–464, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany.

[GHO20] Sanjam Garg, Mohammad Hajiabadi, and Rafail Ostrovsky. Efficient range-trapdoor functions and applications: Rate-1 OT and more. In *TCC 2020: 18th Theory of Cryptography Conference, Part I*, Lecture Notes in Computer Science, pages 88–116. Springer, Heidelberg, Germany, March 2020.

[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.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[IKO$^+$11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 406–425, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th Annual ACM Symposium on Theory of Computing*, pages 20–31, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 669–684, Berlin, Germany, November 4–8, 2013. ACM Press.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, Washington, DC, USA, January 7–9, 2001. ACM-SIAM.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of Paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 678–708, Cham, 2021. Springer International Publishing.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 89–114, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[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.

[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.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

A Laconic Oblivious Transfer from QR

A LOT scheme is a FLOT where the function F is defined as: $F(x_i, (m_0, m_1)) = m_{x_i}$. In [CDG+17] it is shown how to go from LOT to FLOT using garbled circuits. In essence, the sender sends labels using LOT to the receiver and prepares a garbled circuit that implements the function $F_y(\cdot) = F(\cdot, \mathbf{y})$. The receiver can recover $F(\mathbf{D}_i, \mathbf{y})$ by evaluating the garbled circuit.

Moreover, it is known that to build LOT, it is enough to build a hash with encryption scheme, which has compression 2-to-1 [CDG+17].

Definition 16 (Hash with encryption)**.** *Let $L = \mathsf{poly}(\lambda)$. A hash with encryption scheme is a tuple of algorithms:*

- $\mathsf{Setup}(1^\lambda, L)$ *takes as input a securty parameter λ. It outputs a* crs.
- $\mathsf{H}(\mathsf{crs}, \mathbf{x} \in \{0,1\}^L)$ *takes as input a* crs *and* $\mathbf{x} \in^L$. *It outputs a hash value h.*
- $\mathsf{Enc}(\mathsf{crs}, h, (i, \beta) \in [L] \times \{0,1\}, m \in \{0,1\})$ *takes as input a* crs*, a hash h, a position $(i, \beta) \in [L] \times \{0,1\}$, and a message m. It outputs a ciphertext* ct.
- $\mathsf{Dec}(\mathsf{crs}, \mathsf{ct}, \mathbf{x})$ *takes as input a* crs*, a ciphertext* ct *and* $\mathbf{x} \in^L$. *It outputs a message m.*

Correctness states that the decryptor is able to recover the message m, if ct is encrypted with respect to (i, β) and $x_i = \beta$.

Security states that $\mathsf{Enc}(\mathsf{crs}, h, (i, 1 - x_i), 0 \in \{0,1\})$ is indistinguishable from $\mathsf{Enc}(\mathsf{crs}, h, (i, 1 - x_i), 1 \in \{0,1\})$, for $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ and $\mathbf{x} \leftarrow \mathsf{H}(\mathsf{crs}, \mathbf{x} \in \{0,1\}^L)$.

This is captured by the following game: For all adversaries $\mathcal{A}$ we have that

$$\Pr\left[b \leftarrow \mathcal{A}_3(\mathsf{st}_3, \mathsf{ct}) : \begin{array}{c} (\mathbf{x}, L, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\lambda) \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, L) \\ (i, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{st}_1, \mathsf{crs}) \\ b \leftarrow\!\!\$ \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, h, (i, 1 - x_i), b) \end{array} \right].$$

We now show how to build this primitive. The construction follows closely the existing constructions [CDG+17, DG17, DGHM18].

Construction 3. *Let $L = \mathsf{poly}(\lambda)$.*

$\mathsf{Setup}(1^\lambda)$:

- *Generate two safe prime numbers P, Q and compute $N = P \cdot Q$. Sample* $\mathbf{G} = \begin{pmatrix} g_{1,0} & \cdots & g_{L,0} \\ g_{1,1} & \cdots & g_{L,0} \end{pmatrix} \leftarrow\!\!\$ \mathbb{QR}_N^{2 \times L}$.
- *Output* $\mathsf{crs} = (N, g, \mathbf{g})$.

$\mathsf{H}(\mathsf{crs}, \mathbf{x} \in \{0,1\}^L)$: *Output* $h \leftarrow \prod_{i=1}^{L} g_{i,x_i}$.

$\mathsf{Enc}(\mathsf{crs}, h, (i, \beta) \in [L] \times \{0,1\}, m \in \{0,1\})$**:**

- *Sample* $s \leftarrow\!\!\$\ [(N-1)/2]$.
- *Compute* $\mathbf{E} \in \mathbb{J}_N^{2\times L}$ *such that*

$$e_{j,b} = \begin{cases} g_{j,b}^s, \text{ if } j \neq i \vee (j,b) = (i,\beta) \\ \perp, \text{ if } (j,b) = (i, 1-\beta) \end{cases}$$

- *Compute* $f = h^s \cdot (-1)^m \mod N$.
- *Output* $\mathsf{ct} = (\mathbf{E}, f)$.

$\mathsf{Dec}(\mathsf{crs}, \mathsf{ct}, \mathbf{x})$:

- *Parse* ct *as* $(\mathbf{E}, f)$. *Let* $\mathbf{E} = \begin{pmatrix} e_{1,0} & \dots & e_{L,0} \\ e_{1,1} & \dots & e_{L,1} \end{pmatrix}$
- *Compute* $c = \prod_{i=1}^{L} e_{i,x_i} \mod N$
- *If* $f/c \mod N = 1$ *then output* $m = 0$. *Else if* $f/c \mod N = -1$ *output* $m = 1$.

Correctness holds since if $x_i = \beta$, then Dec has access to $e_{i,\beta}$. Hence $c = \prod_{i=1}^{L} e_{i,x_i} = \prod_{i=1}^{L} g_{i,x_i}^s = h^s \mod N$ and, thus, $f/c \mod N = (-1)^m$.

We now argue security. In the following we recall a useful lemma from [BG10].

Lemma 9 ([BG10])**.** *Let N be a uniformly sampled Blum integer, let $\mathbb{QR}_N$ be the multiplicative group of quadratic residues modulo N with generator g, and let $\ell = \ell(\lambda)$ be a polynomial. If the quadratic residuosity assumption holds with respect to $\mathbb{QR}_N$ then for any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*

$$\left| \begin{array}{c} \Pr\left[b \leftarrow \mathcal{A}_2(\mathbf{g}^r \odot (-1)^{\mathbf{m}}) : \begin{array}{c} \mathbf{g} \leftarrow \mathbb{QR}_N^{\ell} \\ \mathbf{m} \leftarrow \mathcal{A}_1(N, \mathbf{g}) \\ r \leftarrow\!\!\$\ [(N-1)/2] \end{array} \right] - \\ \Pr\left[b \leftarrow \mathcal{A}_2(\mathbf{g}^r) : \begin{array}{c} \mathbf{g} \leftarrow \mathbb{QR}_N^{\ell} \\ \mathbf{m} \leftarrow \mathcal{A}_1(N, \mathbf{g}) \\ r \leftarrow\!\!\$\ [(N-1)/2] \end{array} \right] \end{array} \right| \leq \mathsf{negl}(\lambda)$$

where $\mathbf{m} \in \{0,1\}^{\ell}$ and $\odot$ denotes the componentwise multiplication.

Lemma 10 (Security)**.** *The scheme presented above is a secure hash with encryption scheme under the QR assumption.*

Proof. The reduction starts by receiving $\mathbf{x} \in \{0,1\}^L$ from the adversary. It receives $(g_{1,0}, \dots, g_{L,1})$ from the challenger and forwards it to the adversary as the crs. Upon receiving i from the adversary, the reduction does the following:

- Send $\mathbf{m} = (0, \dots, 0, 1, 0, \dots, 0)$ with a 1 on position $2i + (1 - x_i)$, where $\mathbf{m}$ is the same as in Lemma 9.
- Upon receiving the challenge $(e_{1,0}, \dots, e_{L,1})$, it computes $f = \prod e_{i,x_i} \mod N$ and the matrix $\mathbf{E}$ which is composed by the elementss $e_{j,\beta}$ except that the (i, x_i) position is erased
- It sends $(\mathbf{E}, f)$ to the adversary and outputs whatever it outputs.

Note that $f = \prod e_{i,x_i} = \prod g_{i,x_i}^r (-1)^b$. Moreover, the distribution of $\mathbf{E}$ is the same in the reduction. Hence the advantage of the reduction is exactly the same as the one of the adversary. $\square$