Sublinear Secure Computation from New Assumptions

Elette Boyle[1], Geoffroy Couteau[2], and Pierre Meyer[3]

[1] Reichman University and NTT Research. eboyle@alum.mit.edu
[2] Université Paris Cité, IRIF, CNRS. couteau@irif.fr
[3] Reichman University and Université Paris Cité, IRIF, CNRS. pierre.meyer@irif.fr

Abstract. Secure computation enables mutually distrusting parties to jointly compute a function on their secret inputs, while revealing nothing beyond the function output. A long-running challenge is understanding the required communication complexity of such protocols—in particular, when communication can be *sublinear* in the circuit representation size of the desired function. For certain functions, such as Private Information Retrieval (PIR), this question extends to even sublinearity in the input size.
We develop new techniques expanding the set of computational assumptions for sublinear communication in both settings:

- **Circuit size.** We present sublinear-communication protocols for secure evaluation of general layered circuits, given any 2-round rate-1 batch oblivious transfer (OT) protocol with a particular "decomposability" property. In particular, this condition can be shown to hold for the recent batch OT protocols of (Brakerski et al. Eurocrypt 2022), in turn yielding a new sublinear secure computation feasibility result: from Quadratic Residuosity (QR) together with polynomial-noise-rate Learning Parity with Noise (LPN).
  Our approach constitutes a departure from existing paths toward sublinear secure computation, all based on fully homomorphic encryption or homomorphic secret sharing.
- **Input size.** We construct single-server PIR based on the Computational Diffie-Hellman (CDH) assumption, with *polylogarithmic* communication in the database input size n. Previous constructions from CDH required communication $\Omega(n)$. In hindsight, our construction comprises of a relatively simple combination of existing tools from the literature.

Keywords: Foundations · Private Information Retrieval · Secure Multiparty Computation

Table of Contents

1 Introduction

Secure computation enables mutually distrusting parties to jointly compute a function on their secret inputs, while revealing nothing beyond the function output. We focus on the case of two-party computation with semi-honest (passive) security. Since the seminal feasibility results of the 1980s [Yao86, GMW87, BGW88, CCD88], a major challenge in the area of secure computation has been if and when it is possible to break the "circuit-size barrier." This barrier refers to the fact that all classical techniques for secure computation required a larger amount of communication than the size of a boolean circuit representing the function to be computed. In contrast, insecure computation only requires exchanging the inputs, which are usually considerably smaller than the entire circuit.

Early positive results with sublinear communication either required exponential computation [BFKR91, NN01], or (as discussed later) were limited to very simple functions such as point functions [CGKS95, KO97, CG97] or constant-depth circuits [BI05].

Beyond the circuit-size barrier. This situation changed with the breakthrough result of Gentry [Gen09] on *fully homomorphic encryption* (FHE). FHE is a powerful primitive supporting computation on encrypted data, which can be used to build optimal-communication protocols in the computational setting [DFH12, AJL^{+}12], by having parties perform the desired computation *locally* on encrypted inputs without additional communication. However, despite significant efforts, the set of assumptions under which we know how to build FHE is very narrow. Standard approaches are restricted to lattice-based assumptions, such as Learning With Errors (LWE), and in particular do not include any of the traditional assumptions which were used in the 20th century. Very recent developments in indistinguishability obfuscation imply results based on an alternative (relatively exotic) bundle of assumptions [CLTV15, JLS22].[4]

The work of [BGI16] first showed that secure computation with communication sublinear in the circuit size could also be based on assumptions not known to imply FHE, via a new primitive of *homomorphic secret sharing* (HSS). HSS can be viewed as a relaxation of FHE, where homomorphic evaluation can be distributed among two parties who do not interact with each other. More concretely, from the Decisional Diffie-Hellman (DDH) assumption, [BGI16] constructed a form of HSS for branching programs (including NC^1), implying secure computation for the corresponding function class with asymptotically optimal communication. In turn, this was shown to yield secure computation for general layered circuits[5] of size s with sublinear communication $O(s/\log s)$, by evaluating in $(\log s)$-depth blocks, and communicating only between blocks.

Since then, the HSS-based approach and variations have resulted in sublinear-communication secure protocols from an additional assortment of assumptions. Following the [BGI16] blueprint, the works of [FGJI17, OSY21, RS21] were able to replace the DDH assumption with Decision Composite Residuosity (DCR). The framework was recently abstracted and extended to further algebraic structures, including a class of assumptions based on class groups of imaginary quadratic fields [ADOS22]. In addition, the work of [CM21] built HSS for $\log\log$-depth circuits (yielding $O(s/\log\log s)$ communication secure computation for layered circuits) based on a strong flavor of the Learning Parity with Noise (LPN) assumption: with a small number of samples, but assuming super-polynomial hardness, with inverse-superpolynomial noise rate.

To date, these two approaches—FHE and HSS—still comprise the only known paths to sublinear-communication secure computation for general circuit classes, without resorting to superpolynomial computation or setup assumptions such as correlated randomness [IKM$^+$13, DNNR17, Cou19]. It remains a motivated research agenda not only to continue expanding the set of distinct computational assumptions upon which sublinear secure computation can be built, but additionally of exploring new types of approaches toward this goal.

Private Information Retrieval. As mentioned, one exception to the above treatment is the special case of specific simple functionalities: most prominently, the task of *private information retrieval* (PIR) [CGKS95, KO97]. A (single-server) PIR protocol roughly amounts to a secure computation protocol (with one-sided privacy) for the specific function $f(x, i) = x_i$ with $x \in \{0,1\}^n$ and $i \in [n]$. Unlike the case of general computation (where the communication complexity of the underlying function may be $\Omega(n)$ even without security), PIR can admit secure protocols with communication *sublinear (even polylogarithmic) in the input size.*

For many years, protocols for PIR with polylogarithmic communication in n were known only from the Decisional Composite Residuosity (DCR), Learning with Errors (LWE), or Phi-hiding assumptions [CMS99, Cha04, Lip05, OS07]. More recently, such constructions were achieved from Quadratic Residuosity (QR), or Decisional Diffie-Hellman (DDH) [DGI$^+$19].

1.1 Our Results

We present new approaches and techniques for both of the above settings, ultimately extending the set of computational assumptions under which we can achieve sublinear-communication secure computation protocols.

Our results fall within two primary categories:

– We obtain (slightly) sublinear secure two-party computation for general layered circuits, through a new path of low-communication batch oblivious transfer.
– We explore the specific goal of Private Information Retrieval (PIR), and provide a new construction with polylogarithmic communication based on *Computational* Diffie-Hellman (CDH).

[4] Namely, subexponential security of the combination of: Learning Parity with Noise, plus polynomial-stretch pseudorandom generators in NC^0, plus the Decision Linear assumption on symmetric bilinear groups of prime order [JLS22].

[5] A depth-d circuit is layered if it can be divided into d layers such that any wire connects adjacent layers.

We emphasize that our protocols execute in polynomial runtime, and do not rely on any correlated randomness assumptions.

Sublinear 2PC for layered circuits. We present a new approach toward secure two-party computation protocols for general layered circuits, with communication complexity that scales sublinearly in the circuit size. As opposed to building FHE or HSS, our approach begins with protocols for "batch Oblivious Transfer" with low communication.

Oblivious Transfer (OT) is an atomic functionality in which sender and receiver parties begin with inputs $m_0, m_1 \in \{0,1\}$ and $b \in \{0,1\}$, respectively; at the conclusion the receiver learns the selected message m_b; and neither party learns further information about one another's inputs. OT was shown to be a complete functionality for general secure computation [Kil00], where OT protocol execution(s) take place for each nonlinear gate of the corresponding circuit.

OT protocols are known from a number of standard assumptions, in just two rounds of communication (i.e., one message from receiver to sender, and one message in return); but, the communication complexity for all such solutions is (inherently) significantly larger than the input size. Very recently, it was shown by Brakerski et al. [BBDP22] how to achieve a *batched* version of OT, still in two rounds, and with *rate-1* communication. That is, for a collection of message pairs $(\{m_0^{(i)}, m_1^{(i)}\})_{i\in[\ell]}$ and selection bits $(b^{(i)})_{i\in[\ell]}$, a sender and receiver could perform ℓ parallel batched executions of OT in communication roughly ℓ.

We prove that any such protocol which satisfies an additional *decomposability* property suffices to imply secure computation protocols for general layered circuits with sublinear communication complexity. To define decomposability, consider the communication structure of any 2-round rate-1 batch OT protocol. In the first round, the receiver sends $\ell+o(\ell)$ bits to the sender,[6] somehow encoding its selection bits $b^{(i)}$. In response, the sender performs some computation as a function of its message pairs $\{m_0^{(i)}, m_1^{(i)}\}$, and returns $\ell + o(\ell)$ bits in response, somehow encoding the k selected messages, $m_{b^{(i)}}^{(i)}$. For the constructions of [BBDP22], the sender's message size is just $\ell + \mathsf{polylog}(\ell)$.

We say that the (2-round, rate-1) batch OT protocol is *decomposable* if for any agreed subset $S \subset [\ell]$ of indices, the sender can choose a corresponding subset of $|S| + \mathsf{polylog}(\ell)$ of its return message bits, such that sending this partial sender response reveals *exactly* the corresponding subset of selected messages $(m_{b^{(i)}}^{(i)})_{i\in S}$ to the receiver. Namely, given the partial response, these $|S|$ messages can be recovered, and no information is revealed about $m_{b^{(i)}}^{(i)}$ for $i \notin S$.

Theorem 1 (Sublinear 2PC from Decomposable Batch OT - informal). *Assume existence of 2-round rate-1 batch OT with the above "decomposability" property. Then for any k, we can securely compute layered (synchronous) circuits of depth d and size s using* $\mathsf{poly}(2^{2^k}, s)$ *computation and* $O(2^{2^k} \cdot d \cdot \mathsf{poly}(\lambda) + s/k)$ *communication.*

In particular, for $k = O(\log\log s)$*, we obtain communication* $O(s/\log\log s + d^{1/3} \cdot s^{2(1+\varepsilon)/3} \cdot \mathsf{poly}(\lambda))$*, for an arbitrary small constant* ε*. The latter is sublinear in s whenever* $d = o(s^{1-\varepsilon}/\mathsf{poly}(\lambda))$*,* i.e.*, the circuit is not too "tall and skinny".*

This decomposability property is not simply hypothetical, but rather was inspired by the batch-OT protocols of Brakerski et al. [BBDP22], which we show to satisfy the requirement. At a high level, the sender's message in their protocols consists of an encryption of the selected message bits (computed homomorphically as a function of receiver-sent ciphertexts of its selection bits, together with the message pairs $\{m_0^{(i)}, m_1^{(i)}\}$), compressed *à la* [DGI$^+$19] to rate 1. The resulting rate-1 ciphertexts have the structure of a $\mathsf{polylog}(\ell)$-size "header" string, independent of the messages, together with a *single bit* of information for each encrypted message bit. Decomposability thus follows (almost) directly, by simply omitting those information bits corresponding to encrypted messages the sender wishes to drop (i.e., $[\ell] \setminus S$).[7]

In turn, we obtain the following corollary.

Corollary 2 (Sublinear 2PC from QR+LPN - informal). *The conclusion of Theorem 1 holds based on Quadratic Residuosity (QR) and Learning Parity with Noise (LPN) for any inverse-polynomial noise rate.*

[6] Our construction can actually handle arbitrary *constant* client-to-server upload rate, as long as the sender-to-receiver download rate is 1.

[7] We are of course sweeping details under the rug here, and refer the reader to the main body for a more complete treatment.

Our result is summarized on Table 1, where we also recall the state of the art in sublinear secure computation. We remark that while sublinear $O(s/\log\log s)$-communication protocols were known from a variant of LPN from [CM21], their result must assume superpolynomial hardness of LPN with a small inverse-superpolynomial error rate. In contrast, our result requires only polynomial hardness of LPN, with any inverse-polynomial error rate (as inherited by the construction of [BBDP22]).

We finally mention that this result is also not implied by the constructions of pseudorandom correlation functions (PCF) [BCG+20] from QR+LPN of [OSY21] (or in fact any of the line of work on pseudorandom correlation generators (PCG) [BCG+19b]). While PCG/PCFs enable the generation of large quantities of *random* instances of OT with sublinear communication, the best known approaches for utilizing these random correlations within an actual secure computation protocol require communication that scales linearly with the circuit size.

Table 1: Existing protocols for secure computation with sublinear communication under various assumptions, in the computational setting.

	Assumptions	Circuit class	Sublinearity[1]
[Gen09]	LWE	$\mathsf{P/poly}$	$O(n+m)+\mathsf{poly}(\lambda)$
[BGI16]	DDH	Layered circuits	$O(n+m+s/\log s)$
[OSY21, RS21]	DCR	Layered circuits	$O(n+m+s/\log s)$
[CM21]	superpoly LPN[2]	Layered circuits	$O(n+m+s/\log\log s)$
[ADOS22]	Class groups	Layered circuits	$O(n+m+s/\log s)$
This work	LPN + QR[3]	Layered circuits	$O\left(n+m+d^{1/3}\cdot s^{2(1+\varepsilon)/3}\cdot\mathsf{poly}(\lambda)+\frac{s}{\log\log s}\right)$

[1] We use n for input size, m for output size, s for circuit size, and d for circuit depth.
[2] [CM21] assumes the superpolynomial hardness of the LPN assumption with dimension N, $O(N)$ samples, and noise rate $N^{o(1)-1}$.
[3] We assume the polynomial hardness of LPN with dimension N, $\mathsf{poly}(N)$ samples, and inverse-polynomial noise rate.

Private Information Retrieval. Motivated by the goal of building decomposable rate-1 batch OT from new assumptions, we then turn to a deeper exploration of one of the required underlying components from the [BBDP22] batch OT construction: (single-server) Private Information Retrieval (PIR).

We succeed in constructing PIR with polylogarithmic communication from the *Computational Diffie-Hellman* assumption. While this is only one sub-component required to obtain the necessary batch OT from LPN+CDH,[8] this provides one step toward this direction. But, more importantly, it constitutes a new feasibility result of its own right. From CDH, previously no PIR protocol was known with communication $o(n)$.

Theorem 3 (PIR from CDH - informal). *Based on the Computational Diffie-Hellman (CDH) assumption, there exists single-server PIR on n-bit databases with communication* $\mathsf{polylog}(n)$ *and* $\mathcal{O}(\log(n))$ *rounds.*

In hindsight, our construction forms a surprisingly simple and clean combination of two existing tools from the literature. Along the way, we identify an improved procedure for converting between a weak form of "semi-PIR" as considered in [BIP18], which reveals the client's queried index with some probability, to full-blown secure PIR. We refer the reader to the Technical Overview for more details.

2 Preliminaries

Throughout the paper, $[\vec{v}]_I$ denotes the subvector of $\vec{v}$ induced by set of indices I.

[8] Indeed, the approach of [BBDP22] requires also a form of homomorphic encryption compressible to rate 1.

2.1 Quadratic Residuosity Assumption (QR)

We say that N is a Blum integer if $N = p \cdot q$ for some primes p and q such that $p \pmod 4 \equiv q \pmod 4 \equiv 3$. We denote by $\mathbb{J}_N$ the multiplicative group of the elements in $\mathbb{Z}_N^\star$ 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 that $\mathbb{QR}_N$ and $\mathbb{J}_N$ have order $\frac{\phi(N)}{4}$ and $\frac{\phi(N)}{2}$ respectively, where $\phi(\cdot)$ is Euler's totient function. It is useful to write $\mathbb{J}_N : \mathbb{H} \times \mathbb{QR}_N$, where $\mathbb{H}$ is the multiplicative group $(\pm 1, \cdot)$ of order 2. Note that is N is a Blum integer then $\gcd(2, \frac{\phi(N)}{4}) = 1$ and $-1 \in \mathbb{J}_N \setminus \mathbb{QR}_N$.

Definition 4 (Quadratic Residuosity Assumption, [GM82]). *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 p.p.t. adversary $\mathcal{A}$*

$$|\Pr_{a \xleftarrow{\$} \mathbb{QR}_N}[\mathcal{A}(N, g, a) = 1] - \Pr_{a \xleftarrow{\$} \mathbb{QR}_N}[\mathcal{A}(N, g, (-1) \cdot a) = 1]| \leq \mathrm{negl}(\lambda).$$

2.2 Learning Parity with Noise (LPN)

Our constructions rely on the Learning Parity with Noise assumption [BFKL94] (LPN) over $\mathbb{F}_2$ (which is the most standard variant of LPN, but other fields can be considered). Unlike the LWE assumption, in LPN the noise is assumed to have a small Hamming weight. Concretely, the noise is 1 in a small fraction of the coordinates and 0 elsewhere. $\mathsf{Ber}_r(\mathbb{F}_2)$ denote the distribution which outputs 1 with probability r, and 0 with probability $1 - r$.

Definition 5 (LPN). *For dimension $k = k(\lambda)$, number of samples (or block length) $q = q(\lambda)$, noise rate $r = r(\lambda)$, the $\mathbb{F}_2$-$\mathsf{LPN}(k, q, r)$ assumption states that*

$$\begin{aligned} &\{(A, \vec{b}) \mid A \xleftarrow{\$} \mathbb{F}^{q \times k}, \vec{e} \xleftarrow{\$} \mathsf{Ber}_r(\mathbb{F}_2)^q, \vec{s} \xleftarrow{\$} \mathbb{F}_2^k, \vec{b} \leftarrow A \cdot \vec{s} + \vec{e}\} \\ \overset{c}{\approx} &\{(A, \vec{b}) \mid A \xleftarrow{\$} \mathbb{F}_2^{q \times k}, \vec{b} \xleftarrow{\$} \mathbb{F}_2^q\} \end{aligned}$$

Here and in the following, all parameters are functions of the security parameter λ and computational indistinguishability is defined with respect to λ. Note that the search LPN problem, of finding the vector can be reduced to the decisional LPN assumption [BFKL94, AIK09].

2.3 Diffie-Hellman (DDH and CDH)

Definition 6 (Computational Diffie-Hellman). *Let $\mathcal{G}(\lambda)$ be an algorithm that outputs $(\mathbb{G}, p, g)$ where $\mathbb{G}$ is a group of prime order p and g is a generator of the group. The* CDH *assumption holds for generator $\mathcal{G}$ if there exists a negligible function ϵ such that for all PPT adversaries $\mathcal{A}$ the following holds:*

$$\Pr\left[g^{ab} \xleftarrow{\$} \mathcal{A}(\mathbb{G}, p, g, g^a, g^b) \colon \begin{array}{c} (\mathbb{G}, p, g) \xleftarrow{\$} \mathcal{G}(\lambda) \\ a, b \xleftarrow{\$} \mathbb{Z}_p \end{array}\right] \leq \epsilon(\lambda).$$

Definition 7 (Decisional Diffie-Hellman). *We say that the Decisional Diffie-Hellman assumption (*DDH*) holds if there exists a PPT group generator $\mathcal{IG}$ with the following properties. The output of $\mathcal{IG}(1^\lambda)$ is a pair $(\mathbb{G}, g)$ where $\mathbb{G}$ describes a cyclic group of a prime order q (where we use multiplicative notations for the group operation) and g describes a group generator. We assume that q is included in the group description $\mathbb{G}$. We also assume the existence of an efficient algorithm that given $\mathbb{G}$ and descriptions of group elements h_1, h_2 outputs a description of $h_1 h_2$. Finally, we require that for every nonuniform polynomial-time algorithm $\mathcal{A}$ there is a negligible function ϵ such that:*

$$\begin{aligned} &|\Pr[\mathcal{A}(\mathbb{G}, g, g^a, g^b, g^{ab}) = 1 \colon (\mathbb{G}, g) \xleftarrow{\$} \mathcal{IG}; (a, b) \xleftarrow{\$} \mathbb{Z}_q^2] - \\ &\qquad \Pr[\mathcal{A}(\mathbb{G}, g, g^a, g^b, g^c) = 1 \colon (\mathbb{G}, g) \xleftarrow{\$} \mathcal{IG}; (a, b, c) \xleftarrow{\$} \mathbb{Z}_q^3]| \leq \epsilon(\lambda). \end{aligned}$$

3 Technical Overview

3.1 Sublinear 2PC for Layered Circuits from Decomposable Batch OT

We consider Boolean circuits over any base of gates with fan-in two.

Toward our sublinear 2PC result for layered circuits, we begin by focusing on circuits of low depth k (e.g., think of $k = \log\log\log s$), and devise a secure protocol with communication $n+m+(2^{2^k} \cdot \mathsf{poly}(\lambda))$, for input size n, output size m, circuit size s, and security parameter λ. Given such a tool, we can appropriately divide a larger layered circuit into depth-k blocks where the sum of all block input and output sizes is s/k, and then iteratively compute (secret shares of) each layer output via the sub-protocol. Combined, this yields a secure computation for the layered circuit with overall communication $O(s/k + 2^{2^k} \cdot d \cdot \mathsf{poly}(\lambda))$, as desired.

Starting point: An SPIR viewpoint. Consider a circuit with input size n, output size m, and low depth k. Given fan-in 2, each output bit is computed as a function of at most 2^k input bits. We may thus view the circuit output as dictated by m separate truth tables, each of size 2^{2^k}, indexed by the values of the corresponding relevant 2^k input bits. More concretely, think of one party as holding the (partially collapsed) truth tables incorporating its known inputs, and the second party as holding its own input string, dictating the relevant position of each truth table. We will refer to the first party as "sender" and second as "receiver."

Given this perspective, protocols for (Symmetric) Private Information Retrieval (SPIR) immediately come to mind. An SPIR protocol is a strengthened version of PIR, where the client additionally learns nothing beyond its queried value of the database. Secure computation of our circuit precisely amounts to m instances of SPIR, where the receiver party learns exactly the desired indexed values of the m truth tables.

However, the situation is not so simple: Even the best known (S)PIR protocols have communication polylogarithmic in the database size. Applying m instances of SPIR for the m outputs would thus yield communication $\mathsf{polylog}(2^k) \cdot m \in \Omega(km)$, killing sublinearity.

In order to obtain sublinear communication, we must somehow leverage that the m SPIR instances are not completely independent, but rather are made with *correlated* queries. That is, although there are m instances each with (2^k)-bit index values, the $m \cdot 2^k$ selection bits have several repeats, collectively coming from different subsets of only $n < m \cdot 2^k$ input bits.

Toward batch SPIR with correlated queries. Our task becomes precisely to construct such an object: m-instance batch SPIR, with significantly lower communication complexity given correlated queries.

For purposes of discussion, suppose there existed a 2-round rate-1 protocol for oblivious transfer, where each sender and receiver (magically) sends only a single bit. Given access to such a tool, then by leveraging ideas from the literature (e.g., achieving PIR from linearly homomorphic encryption [KO97]), we would be set. Indeed, the receiver would simply send 1 bit for each input bit, corresponding to the first OT message using this value as a selector bit. These first messages could then be *reused* by the sender in multiple, recursive executions.

More concretely, suppose the server holds a database of N bits and that the receiver wants to retrieve the element stored at index $x = (x_1, \ldots, x_{\log N})$. If the receiver sends a message otr_1 generated as its first-round OT-receiver message for the first bit x_1 of the desired index, the server can take the database, pair up elements whose indices differ only on the first bit, then apply the OT-server computation with respect to otr_1 on each pair in order to retrieve a single-bit response for each, creating a new "database" of half the number of elements, each corresponding to a 1-bit sender answer message. If instead the receiver sends messages $(\mathsf{otr}_1, \ldots, \mathsf{otr}_{\log N})$, one for each bit of the desired index, the server can now iteratively compress the database down to a single bit by building a "Merkle tree" where in each recursive iteration corresponding to input index bit x_i, the new "database" is split into pairs of messages whose indices differ only in this index, and performing the OT-server computation on each pair produces a new list of 1-bit sender answer messages of again half the length. At the conclusion, the server will be left with a single message value remaining, which by construction precisely enables the receiver to recover the target value stored at index x. This approach extends directly for m distinct databases with the *same* total receiver message $(\mathsf{otr}_1, \ldots, \mathsf{otr}_{\log N})$, since the corresponding OT-receiver messages can be used independently in any mix and match format across databases. In turn, the sender would need to send only m total bits response, one bit for each database query.

Of course, unfortunately, we do not have such a strong rate-1 OT. We thus turn to the next closest alternative which does exist: 2-round rate-1 *batch* OT, as recently achieved by Brakerski et al. [BBDP22]. Batch OT considers a collection of ℓ message pairs $(\{m_0^{(i)}, m_1^{(i)}\})_{i\in[\ell]}$ and selection bits $(b^{(i)})_{i\in[\ell]}$, and enables a sender and receiver to perform ℓ parallel batched executions of OT with communication roughly ℓ. Attempting to apply the above strategy with rate-1 batch OT, however, poses significant challenges.

– The batching structure restricts the "mix and match" abilities of the sender when using the receiver's OT message. The sender must respond to the *entire* batched vector of receiver's selection bits at any stage, without freely accessing subsets of selection bits. Instead, the above approach involves using each selection bit $b^{(i)}$ within a *different* number $(N/2^i)$ of message pairs.
– Even worse, the sender's (batch) response in general is only defined given *all ℓ pairs of messages* to be selected by the bits $b^{(1)}, \ldots, b^{(\ell)}$. In contrast, the above approach crucially relies on the ability to choose the message pairs for selection bit $b^{(i)}$ *dynamically* as a function of the server's responses given the previous selection bits $b^{(1)}, \ldots, b^{(i-1)}$.
– Finally, it is no longer the case that for each selected message the sender has a *single* corresponding response bit. In fact, rate 1 here does not even mean that for ℓ instances that exactly ℓ bits are sent in each direction, but rather just asymptotically $\ell + o(\ell)$. This means that in each recursive OT execution, the sender's messages (and thus "database entry" size) may grow, leading to large growth and ultimately large communication upon further recursions.

Decomposable batch OT. With this motivation, we introduce the notion of *decomposable (2-round, rate-1) batch oblivious transfer*, which can be seen as a strengthening of two-round batch OT with constant upload-rate (*i.e.* the size of the receiver message is linear in the batch size ℓ) and download-rate asymptotically one (*i.e.* the size of the sender message is $\ell + o(\ell)$). The differences boil down to a notion of *decomposability* which we impose on the sender message.

At a high level, what we want to capture is the fact that the receiver should be able to retrieve the i^{th} selected message in the batch if and only it also has access to the i^{th} bit of the sender message (using its own internal state saved from generating the receiver message). More generally, given only a subset of the bits of the sender message, the receiver should able to retrieve the corresponding subset of selected messages in the batch.

Slightly more formally, we say that the (2-round, rate-1) batch OT protocol is *decomposable* if for any agreed subset $S \subset [\ell]$ of indices, the sender can choose a corresponding subset of $|S| + \mathsf{polylog}(\ell)$ of its return message bits, such that sending this partial sender response reveals *exactly* the corresponding subset of selected messages $(m_{b^{(i)}}^{(i)})_{i\in S}$ to the receiver. Namely, given the partial response, these $|S|$ messages can be recovered, and no information is revealed about $m_{b^{(i)}}^{(i)}$ for $i \notin S$.

For our purposes, it will suffice to consider a relaxation of the notion we just described, and allow the sender message to have some small overhead rather than having a one-to-one correspondence between the bits on the sender message and the ℓ selected messages. In this relaxed form, we require that the sender message be comprised of two parts: a "reusable" part (of size $o(\ell)$), and a "decomposable" part (of size ℓ). On its own, the reusable part should reveal nothing about the messages, but can be used to "decode" each bit of the decomposable part so as to retrieve (exactly) the corresponding selected message in the batch. Among other benefits of this relaxation, it allows us to consider constructions whose download-rate is only asymptotically one.

This decomposability property is not only enough for our needs, but perhaps more importantly, is *achievable*, in fact achieved by the batch OT constructions of [BBDP22]. Roughly speaking, the sender message in their construction is composed of a rate-1 encryption of the vector of requested message bits, with structure consisting of a short "header" independent of the message bits, together with a single ciphertext bit encoding each message bit separately. Decomposability can then be achieved by sending only those ciphertext bits encoding the desired subset of messages.

Slightly more accurately, this describes the situation for all but an inverse-polynomial fraction of message bits (corresponding to noisy coordinates of an LPN ciphertext sent by the receiver), which actually encode the *incorrect* messages. In order to separately address these values, they employ a "co-PIR" (or "punctured OT" [BGI17]) to efficiently mask out the undesired values from the receiver, and a separate PIR to learn the correct values for these positions. The separate PIR query responses appear as part of the short "header" information of the server's response, which may sound like an issue, as this portion should not reveal information directly about any message bits. However, this problem does not occur, because the extra PIR queries are set up to actually reveal the *difference* between the masked-out incorrect message $(r_i \oplus m_{1-b})$ and the target message m_b. Because of the mask, this difference value (revealed in the header) provides no information about any message in the absence of the corresponding value $(r_i \oplus m_{1-b})$ from the payload portion of the ciphertext, as required by decomposability. We refer the reader to Section 4.2 for further details.

Sublinear 2PC from decomposable batch OT. This decomposability property directly allows us to address one of the above challenges of batch OT: we will not have issues with exponential growth of the database entry size in the recursive OT executions. Instead, the result of one iteration of the batch OT on n inputs will result in a short $o(n)$-size header together with n bits that each provide information about a distinct queried message. The header string we will put to the side (ultimately we will send the collection of all the headers, which is still sufficiently short). The remaining n bits induce the recursive sender-message database that, as desired, consists of exactly 1 bit per message.

In fact, if we temporarily suppose that the assignment graph structure of n input bits to $m = n$ output bits can be decomposed as the disjoint union of 2^k matchings, then we have a solution. Each disjoint matching will correspond directly to a different instance of n-input batch OT, where each of the n inputs is simultaneously used to index a different database. Applying the recursive solution as above, the sender will ultimately compute a single bit for each output, as well as a collection of header strings from each of the batch OT executions.

The remaining challenge is that general circuits do *not* have such nice regular structure, instead with inputs appearing in different numbers of output computations, with inconvenient correlations, demanding a stronger form of "mix and match" of batched OT queries beyond a direct approach.

To address this issue, we modify the structure of batch OT receiver queries, effectively extending the batch size (say from n to $2n$), and employing a careful choice of how to pack extra copies of more highly influential input bits into the queried vector, so that the overall total number of batch OT instances remains sufficiently small that the overhead of extra header strings does not negatively impact the final communication complexity. We refer the reader to the technical body for a detailed treatment of this procedure.

3.2 Polylogarithmic PIR from CDH

We now turn our attention to our second contribution: private information retrieval with polylogarithmic communication from the computational Diffie-Hellman assumption. A private information retrieval (PIR) is a two party protocol between a server S holding a string z (the database) and a client C holding an integer i. At the end of the interaction, the client should learn z_i, without revealing i to the server. A polylogarithmic PIR is a PIR where the total communication is $\mathsf{poly}(\lambda, \log |z|)$, where λ is the security parameter.

Below, we sketch our approach to building polylogarithmic PIR from CDH. In hindsight, our construction is in fact relatively straightforward, and follows from an elegant combination of two recent results. We outline the sequence of implications below.

$$\begin{tikzcd}
\boxed{\text{CDH}} \arrow[r, Rightarrow, "\text{[ABD}^+\text{21]}"] & \boxed{\text{Laconic PSI}} \arrow[r, Rightarrow, "\text{Lemma 27}"] & \boxed{\text{Half-PIR}} \arrow[r, Rightarrow, "\text{Lemma 30}"] & \boxed{\begin{array}{c}\text{Random-index}\\ \text{PIR}\end{array}} \arrow[r, Rightarrow, "\text{[GHM}^+\text{21]}"] & \boxed{\text{PIR}}
\end{tikzcd}$$

Laconic PSI. A private set intersection protocol is a two-party protocol allowing a receiver to securely compute the intersection of its input set S_R with the set S_S of a sender: at the end of the protocol, the receiver learns $S_R \cap S_S$ and nothing more. A *laconic* PSI protocol, introduced in [ABD$^+$21], additionally enforces that the protocol is two-round (receiver to sender, then sender to receiver), and both the total communication and the sender runtime are bounded by $\mathsf{poly}(\lambda, \log |S_R|, |S_S|)$. The work of [ABD$^+$21] showed that laconic PSI can be constructed from anonymous hash encryption, a primitive that can be constructed (in particular) from the CDH assumption [DG17b,DG17a,BLSV18].

From Laconic PSI to Half-PIR. Given a laconic PSI protocol, we exhibit a construction of polylogarithmic-communication PIR, using in addition a pseudorandom function. However, our construction only achieves a very weak form of security: it only guarantees that the index i is kept hidden from the server with probability 1/2. This notion, which we call half-PIR, has been introduced in [BIP18] (under the name $\mathsf{Rand}\frac{1}{2}\mathsf{PIR}$). It was shown in [BIP18] that polylogarithmic half-PIR already suffices to construct *slightly sublinear* PIR (with communication $O(|z|/\log|z|)$); looking ahead, we will provide a stronger reduction and show that it actually implies polylogarithmic PIR.

Our construction of half-PIR proceeds as follows: the client and the server agree on a PRF key K. The server with input z builds the set $S_R = \{F_K(1||z_1), \cdots, F_K(|z|||z_{|z|})\}$, and the client with input i builds the set $S_S = \{F_K(i||b)\}$, where b is a uniformly random bit. The core properties that this achieves are:

– If $b = z_i$, then $|S_R \cap S_S| = 1$ (note that $|S_S| = 1$), and
– If $b \neq z_i$, then $|S_R \cap S_S| = 0$ with high probability.

To show the second property, we rely on the security of the PRF to argue that a collision between PRF evaluations on distinct inputs is highly unlikely (provided the PRF outputs are large enough). Note, therefore, that we rely on the PRF *security* to argue the *correctness* of the construction (while this is slightly unusual, this kind of arguments has been used a few times in the literature).

Now, the server and the client execute a laconic PSI, which has total communication $\mathsf{poly}(\lambda, \log|z|)$ (since $|S_S| = 1$). At the end of the protocol, the server, who plays the role of the receiver, sends $|S_R \cap S_S|$ to the client. Note that $|S_R \cap S_S| = (1 - b) \oplus z_i$, hence the client can decode z_i from this information. Yet, whenever $|S_R \cap S_S| = 0$, the security of the laconic PSI implies that the server actually learns nothing about i: this guarantees client security with probability 1/2. When $|S_R \cap S_S| = 1$, however, the server learns the intersection $S_R \cap S_S = F_K(i||z_i)$, and can in particular retrieve i easily.

From Half-PIR to PIR. We now turn to constructing a polylogarithmic PIR from a polylogarithmic half-PIR. Here, our construction is mostly a simple observation: half-PIR implies random-index PIR via a straightforward construction. A random-index PIR, introduced in [GHM+21], is a PIR protocol where the client has no input, and receives (i, z_i) where the index i is picked uniformly at random between 1 and $|z|$. Given a half-PIR, building a random-index PIR is almost immediate: the client and the server execute λ parallel instances of a half-PIR protocol, where the client uses uniformly random independent indices in each instance. With overwhelming probability, at least one of these instances will be secure (in the sense that the server does not learn the index); the client simply outputs (i^*, z_{i^*}) where i^* is the index used in the first such execution.

Eventually, random-index PIR was recently shown in [GHM+21] to imply full-fledged PIR, with a $\log|z|$ blowup in communication and round complexity. The key observation underlying this reduction is that a single invocation of a random-index PIR, together with sending $\log|z|$ bits, allows to reduce the task of executing a PIR on a size-$|z|$ database to that of executing a PIR on a size-$|z|/2$ database. The construction follows by recursively invoking this construction (we provide a more detailed description of this construction in Section 5.3). Combining all these building blocks together leads to a logarithmic-round, polylogarithmic-communication PIR from the CDH assumption.

4 Sublinear Computation for loglog-Depth Circuits

4.1 Decomposable Two-Round Batch Oblivious Transfer

In this section, definition 8 defines the notion of *decomposable two-round batch oblivious transfer* (dec-OT).
We introduce the notion of *decomposable two-round batch oblivious transfer*, which can be seen as a strengthening of two-round batch OT with constant upload-rate (*i.e.* the size of the receiver message is linear in the batch size k) and download-rate asymptotically one (*i.e.* the size of the sender message is $k + o(k)$). The differences boil down to a notion of *decomposability* which we impose on the sender message. At a high level, what we want to capture is the fact that the receiver should be able to retrieve the i^{th} selected message in the batch if and only it also has access to the i^{th} bit of the sender message (using its own internal state saved from generating the receiver message). More generally, given only a subset of the bits of the sender message, the receiver should able to retrieve the corresponding subset

of selected messages in the batch. For our purposes, it will suffice to consider a relaxation of the notion we just described, and allow the sender message to have some small overhead rather than having a one-to-one correspondence between the bits on the sender message and the k selected messages. In this relaxed form, we require that the sender message be comprised of two parts: a "reusable" one (of size $o(k)$), and a "decomposable" one (of size k). On its own, the reusable part should reveal nothing about the messages, but can be used to "decode" each bit of the decomposable part so as to retrieve (exactly) the corresponding selected message in the batch. Among other benefits of this relaxation, it allows us to consider constructions whose download-rate is only asymptotically, and not necessarily exactly, optimal. We now formalize this notion in definition 8.

Definition 8 (Decomposable Two-Round Batch Oblivious Transfer). *Let $k \in \mathbb{N}^\star$ and let $\alpha(\cdot)$ be a sublinear function (*i.e. $\alpha(n) = o(n)$*). A semi-honest two-round decomposable batch OT protocol with $\alpha(\cdot)$-overhead between a sender and a receiver is defined as a triple of PPT algorithms* $\mathsf{dec\text{-}OT} = (\mathsf{dec\text{-}OTR}, \mathsf{dec\text{-}OTS}, \mathsf{dec\text{-}OTD})$ *with the following syntax and properties:*

- **Syntax.**
  - $\mathsf{dec\text{-}OTR}$: *On input the security parameter 1^λ and a vector of selection bits $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$,* $\mathsf{dec\text{-}OTR}$ *outputs a receiver message* $\mathsf{otr} \in \{0,1\}^{\mathcal{O}(k)}$ *and an internal state* st*; without loss of generality we assume that* st *contains all the random coins used by* $\mathsf{dec\text{-}OTR}$ *as well as $\vec{b}$.*
  - $\mathsf{dec\text{-}OTS}$: *On input the security parameter 1^λ, a receiver message* otr*, and a database of k pairs of bits $((m_0^{(i)}, m_1^{(i)}))_{i\in[k]} \in \{0,1\}^{2k}$,* $\mathsf{dec\text{-}OTS}$ *outputs a sender message* $\mathsf{ots} = (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}})$*, which is comprised of a reusable part* $\mathsf{ots}^\star \in \{0,1\}^{\alpha(k)}$ *and a decomposable part* $\mathsf{ots}^{\mathsf{dec}} \in \{0,1\}^k$.
  - $\mathsf{dec\text{-}OTD}$: *On input a batch subset $K \subseteq [k]$, a partial sender message* $\mathsf{ots}' \in \{0,1\}^{\alpha(k)+|K|}$*, and an internal state* st*,* $\mathsf{dec\text{-}OTD}$ *outputs a vector of messages $(\widetilde{m}_i)_{i\in K} \in \{0,1\}^{|K|}$.*
- **Decomposable Correctness.** *For every $\lambda \in \mathbb{N}^\star$, $K \subseteq [k]$, every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, and every $\vec{m} = ((m_0^{(i)}, m_1^{(i)}))_{i\in[k]} \in \{0,1\}^{2k}$,*

$$\Pr\left[(\widetilde{m}_1, \ldots, \widetilde{m}_{|K|}) = (m_{b_i}^{(i)})_{i\in K} : \begin{array}{r} (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{\mathit{DB}}) \xleftarrow{\$} \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}) \\ (\widetilde{m}_1, \ldots, \widetilde{m}_{|K|}) \xleftarrow{\$} \mathsf{dec\text{-}OTD}(K, (\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K), \mathsf{st}) \end{array}\right] = 1 .$$

- **Receiver Security (against Semi-Honest Sender).** *There exists an expected polynomial time simulator* Sim_S *such that for every $\lambda \in \mathbb{N}^\star$ and every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$,*

$$\left\{\mathsf{otr} \colon (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b})\right\} \overset{c}{\approx} \left\{\mathsf{Sim}_S(1^\lambda)\right\} .$$

- **Decomposable Sender Security (against Semi-Honest Receiver).** *There exists an expected polynomial time simulator* Sim_R *such that for every $\lambda \in \mathbb{N}^\star$, every $K \subseteq [k]$, every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, and every $\vec{m} = ((m_0^{(i)}, m_1^{(i)}))_{i\in[k]} \in \{0,1\}^{2k}$,*

$$\left\{(\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K, \mathsf{otr}, \mathsf{st}) \colon \begin{array}{r} (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{\mathit{BD}}) \xleftarrow{\$} \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}) \end{array}\right\} \overset{c}{\approx}$$

$$\left\{(\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}, \mathsf{otr}, \mathsf{st}) \colon \begin{array}{r} (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{Sim}_R(1^\lambda, K, (m_{b_i}^{(i)})_{i\in K}, \vec{b}, \mathsf{otr}, \mathsf{st}) \end{array}\right\} .$$

Note that in definition 8 the rate is baked into the syntax: $\mathsf{dec\text{-}OTR}$ outputs a receiver message of size $\mathcal{O}(k)$ and $\mathsf{dec\text{-}OTS}$ outputs a sender message of size $\alpha(k) + k$.

4.2 Instantiation under $\mathsf{QR} + \mathsf{LPN}$, Adapted from [BBDP22]

As noted previously, two-round decomposable batch oblivious transfer can be seen as a strengthening of two-round batch OT with constant upload-rate and download-rate asymptotically one. As a matter of fact, the construction of batch OT with optimal rate from [BBDP22] natively satisfies the extra requirements and can be cast as two-round decomposable batch OT with sublinear overhead.

Theorem 9 (Corollary of [BBDP22, Section 7]). *Assume the* QR *assumption and the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim}, \mathsf{num}, \rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = \mathsf{dim}^c$ *(for any constant* $c > 1$*), and noise rate* $\rho = \mathsf{num}^{\varepsilon - 1}$ *(for some constant* $\varepsilon < 1$*). Then for any* $\ell = \ell(\lambda)$*, there exists a decomposable two-round batch oblivious transfer for batch size* $k = \ell \cdot \mathsf{num}$ *where*

- *The receiver message* otr *has size* $(\ell^2 \cdot \mathsf{dim} + \ell \cdot \mathsf{num}^\varepsilon) \cdot \mathsf{poly}(\lambda) + k$
- *The sender message* $\mathsf{ots} = (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}})$ *has size* $|\mathsf{ots}^\star| = (\mathsf{num} + \ell \cdot \mathsf{num}^\varepsilon) \cdot \mathsf{poly}(\lambda)$ *and* $|\mathsf{ots}^{\mathsf{dec}}| = k$.

In particular, for appropriate parameters (sufficiently large ℓ*, and* num *sufficiently larger than* ℓ*),* $|\mathsf{otr}| = k + o(k)$*, and* $|\mathsf{ots}^\star| = o(k)$*.*

What follows is a sketch of how to cast the constructions of two-round rate-one batch OT from [BBDP22] as decomposable, thereby proving theorem 9. We refer to appendix A for more details.

Sketch. The size of the OT messages can be directly read from the analysis of the communication complexity of the construction in [BBDP22, Section 7.1] (we let the $\mathsf{poly}(\lambda)$ term absorb a $\mathsf{polylog}(\mathsf{num})$ term). Receiver security is directly proven in [BBDP22, Section 7.2].

To show decomposable correctness, we must prove that given a subset $K \subseteq [k]$ of the bits of $\mathsf{ots}^{\mathsf{dec}}$, it is still possible to recover the messages $(m^i_{b_i})_{i \in K}$. Abstracting out the unecessary details, the output algorithm $\mathsf{dec\text{-}OTD}$ will decrypt a matrix $\tilde{W} \in \mathbb{F}_2^{\ell \times \mathsf{dim}}$ from ciphertexts contained in ots. The entries of this matrix are $\ell \cdot \mathsf{dim}$ outputs $\tilde{m}^{(i)}_{b_i}$, each masked with some pseudorandom value. The intuition is that for most entries, $\tilde{m}^{(i)}_{b_i}$ is just the right output $m^{(i)}_{b_i}$. However, for some entries, $\tilde{m}^{(i)}_{b_i}$ is incorrect (*i.e.*, noisy): this is a consequence of the imperfect correctness of the LPN-based encoding used in the construction. Then, a second matrix Z will be obtained, which allows to unmask the correct entries of $\tilde{W}$, and correct the noisy entries. The final output is given by the matrix $\tilde{W} + Z$. This matrix Z is obtained using a combination of a private information retrieval scheme with polylogarithmic communication (which allows to recover the small subset of noisy entries) with a co-PIR, a primitive which allows to recover all pseudorandom masks except for a small subset of them, using small communication. Summing up, there are two components:

1. The matrix Z is extracted from $\mathsf{ots}^\star$ using the state information stored after $\mathsf{dec\text{-}OTR}$. Crucially, Z is entirely obtained from $\mathsf{ots}^\star$: $\mathsf{ots}^{\mathsf{dec}}$ is not required at this stage.
2. The matrix $\tilde{W}$ is extracted from ots by decrypting a ciphertext contained in the OT message. The output is $\tilde{W} + Z$.

The second part is the only one that requires $\mathsf{ots}^{\mathsf{dec}}$. In the construction of [BBDP22], $\mathsf{ots}^{\mathsf{dec}}$ contains the *shrinked* components of a rate-1 linearly homomorphic encryption scheme. Using the QR-based construction of rate-1 linearly homomorphic encryption from [BBDP22, Section 5.3], which is itself taken from [DGI$^+$19]. In this scheme, an encryption of $b_1, \cdots, b_\ell$ is of the form

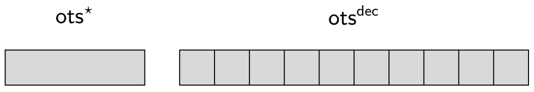
$$\mathsf{ct} = (g^r, (-1)^{b_1} h_1^r, \cdots, (-1)^{b_\ell} h_\ell^r),$$

where $g, h_1, \cdots, h_\ell$ are random quadratic residues in an RSA group, and r is a random exponent. The shrinking procedure maps $c_i = (-1)^{b_i} h_i^r$ to $b'_i = 1$ if $c_i < -c_i$ (given some order over $\mathbb{J}_n$, the set of elements with Jacobi symbol 1), and 0 otherwise. The crucial observation is that given g^r (which, for us, will be in the $\mathsf{ots}^\star$ part) and any given individual shrinked component b'_i, it is possible to "decompress" b'_i into c_i, by computing h_i^r from g^r (using the secret key) and checking whether $h_i^r < -h_i^r$. This implies, in our setting, that the i-th entry of $\tilde{W}$ can be individually decrypted (using the secret key, stored as part of the state), given $\mathsf{ots}^\star$ and the i-th bit of $\mathsf{ots}^{\mathsf{dec}}$. This guarantees decomposable correctness.

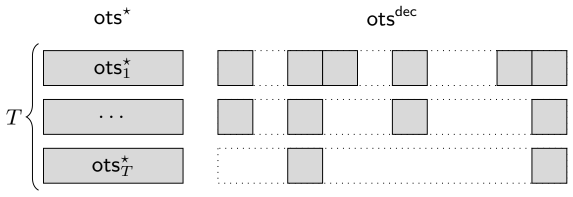
Eventually, it remains to show why decomposable sender security holds; again, it follows almost immediatly from the sender security of the construction in [BBDP22]. The only thing to show is that the simulator Sim can still simulate the *partial* OT messages $(\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K)$ when it is given only the subset $(m^i_{b_i})_{i \in K}$ of the sender messages. But this is completely straightforward, since (as implied by the discussion on correctness above) the distribution of $(\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K)$ is essentially identical to the distribution of $(\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}})$ obtained given a size-$|K|$ sender input $(m^i_0, m^i_1)_{i \in K}$ and a size-$|K|$ receiver input $(b_i)_{i \in K}$ (the only –minor– distinction being that the $\mathsf{ots}^\star$ contains some "dummy" g^r components which are not used anywhere, but this has no influence on the simulation). This concludes the sketch. □

4.3 Bounded Query Repetitions

At a high level the goal of this section is to show how a receiver message of dec-OT can be re-used, possibly with imbalances in how many times each selection bit in the batch is re-used, while asymptotically preserving upload- and download-rate. definition 10 defines the notion of *decomposable two-round batch oblivious transfer with bounded query repetitions* (rep-OT), and lemma 11 establishes a generic reduction from dec-OT to rep-OT, with an explicit transformation in fig. 3.
A central property of two-round OT protocols is that the receiver message can be re-used multiple times by the sender on different (and even adaptively crafted) databases. If the receiver were to prepare messages for different selection bits, the sender could use each one a different number of times and still only have to send an answer whose size is proportional to the number of "useful" messages. When using two-round batch OT, the sender cannot freely re-use some of the selection bits more than the others: if the receiver sends a message corresponding to the selection vector $(b_1, \dots, b_k)$, and the sender wishes to use each selection bit b_i a number t_i of times then it has little choice but to pad their database up to size $(\max_{i\in[k]} t_i) \times k$ using dummy messages then send $(\max_{i\in[k]} t_i)$ different sender messages. However, if the two-round batch OT is decomposable, then the sender can drop the bits corresponding to the dummy messages before sending them to the receiver. More specifically, if the two-round decomposable batch OT has α-overhead, then the size of the sender message is $(\max_{i\in[k]} t_i) \cdot \alpha + \sum_{i=1}^{k} t_i$. Since the amount of "useful" bits is $\sum_{i=1}^{k} t_i$, the download rate is still asymptotically one provided α is sufficiently small. This informal discussion is illustrated in fig. 1.



(a) The reusable part ots* is monolithic, while the decomposable part ots$^{\mathsf{dec}}$ can be decomposed.

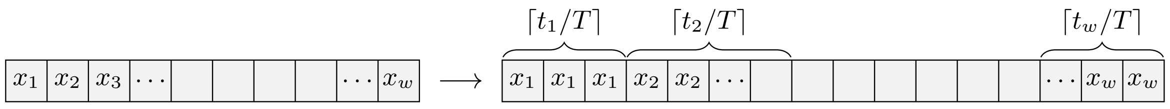


(b) Having invoked T parallel instances of dec-OT, the reusable part is now T times larger, but each selection bit can be used up to T times without the size of the decomposable part blowing up by a factor of T.

Fig. 1: Using decomposability to achieve $\|\cdot\|_\infty$-bounded query repetitions.

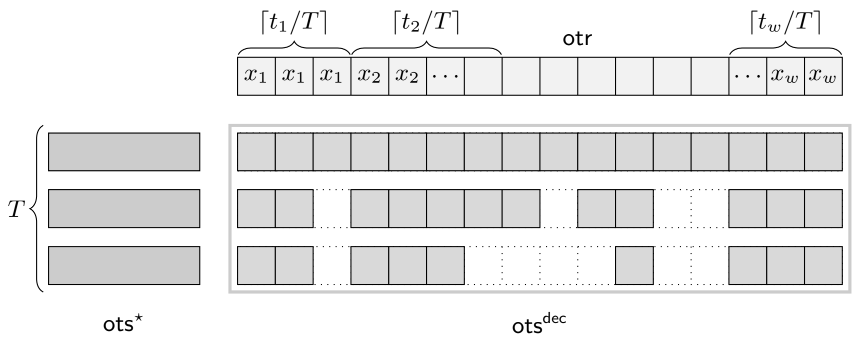
We introduce a stronger notion of two-round decomposable batch OT in definition 10, one which allows each selection bit to be used a different but bounded number of times while preserving the asymptotic rate. We provide in fig. 3 a black-box transformation which allows any two-round decomposable batch OT to gain this property. Before we proceed, let us observe that this transformation has an inherent limitation: because the size of the sender message grows with $(\max_{i\in[k]} t_i) \times \alpha$ the number of repetitions must be bounded by $\|t\|_\infty = o(k/\alpha)$. In particular, none of the t_i's can be linear in k (and skipping ahead, our application to sublinear computation will require us to bypass this restriction). In other words, while we can tolerate $\|\cdot\|_\infty$-bounded repetitions, we cannot tolerate $\|\cdot\|_1$-bounded repetitions; the difference boils down to the fact that we require the repetitions to be somewhat "balanced" in that no selection bit is solicited too much (*e.g.* a linear number of times). This can be addressed by having the receiver replicate the selection bits proportionally to the number of times the sender will want to use it. This way, we are reduced to the case where the repetitions are balanced: if each selection bit b_i is replicated across $\lceil t_i/T \rceil$ copies, then each copy of each selection bits is only used $\leq T$ times, which means the repetition vector is $\|\cdot\|_1$-bounded by T. In the parameter range of our application, T can be chosen so that $T \times \alpha$ is sublinear but the total number of copies sent by the receiver, which is $\sum_{i=1}^{w} \lceil t_i/T \rceil \leq \frac{1}{T}\|t\|_1 + w$, is less than $2w$. What this achieves is keeping the sender message small by making the receiver message only slightly larger. This transformation is illustrated in fig. 2 and implicitly used in the construction of fig. 7.

Definition 10 (Decomposable Two-Round Batch Oblivious Transfer with Bounded Query Repetitions). *Let $k \in \mathbb{N}^\star$ and $\alpha = o(n)$. A semi-honest two-round decomposable batch OT protocol*



$\lceil t_1/T \rceil$
$\lceil t_2/T \rceil$
$\lceil t_w/T \rceil$
x_1 x_2 x_3 $\cdots$ $\cdots$ x_w
x_1 x_1 x_1 x_2 x_2 $\cdots$ $\cdots$ x_w x_w

(a) The overall size of the receiver message with duplicated queries $\Sigma_{i=1}^{w}\lceil t_i/T \rceil \le 2w$.



(b) If $\|t\|_1 = w$ we now have $|\mathsf{otr}| = \mathcal{O}(w)$ and $|\mathsf{ots}| = w \cdot (1 + o(1))$.

Fig. 2: Going from $\|\cdot\|_\infty$-bounded repetitions to $\|\cdot\|_1$-bounded repetitions by doubling the size of the receiver message.

with $\alpha(\cdot)$-overhead and T-bounded query repetitions between a sender and a receiver can be defined as a triple of PPT algorithms $\mathsf{rep\text{-}OT} = (\mathsf{rep\text{-}OTR}, \mathsf{rep\text{-}OTS}, \mathsf{rep\text{-}OTD})$ *with the following syntax and properties:*

– **Syntax.**

$\mathsf{rep\text{-}OTR}$: *On input the security parameter 1^λ and a vector of selection bits $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$,* $\mathsf{rep\text{-}OTR}$ *outputs a receiver message* $\mathsf{otr} \in \{0,1\}^{\mathcal{O}(k)}$ *and an internal state* st*; without loss of generality we assume that* st *contains all the random coins used by* $\mathsf{rep\text{-}OTR}$ *as well as $\vec{b}$.*

$\mathsf{rep\text{-}OTS}$: *On input the security parameter 1^λ, a query* otr*, a database $((m_0^{(i)}, m_1^{(i)}))_{i\in[k']} \in \{0,1\}^{2k'}$ (where $k \le k' \le k \cdot T$), and a vector of repetitions* $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0,T]^k$ *such that $\sum_{i=1}^{k} \mathsf{rep}_i = k'$,* $\mathsf{rep\text{-}OTS}$ *outputs a sender message* $\mathsf{ots} = (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}})$*, which is comprised of a reusable part* $\mathsf{ots}^\star \in \{0,1\}^{\alpha(k)}$ *and a decomposable part* $\mathsf{ots}^{\mathsf{dec}} \in \{0,1\}^{k'}$*, as well as* rep.

$\mathsf{rep\text{-}OTD}$: *On input a batch subset $K \subseteq [k']$, a partial sender message* $\mathsf{ots}' \in \{0,1\}^{\alpha(k)+|K|}$*, a vector of repetitions* $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0,T]^k$ *such that $\sum_{i=1}^{k} \mathsf{rep}_i = k'$, and an internal state* st*,* $\mathsf{rep\text{-}OTD}$ *outputs a vector of messages $(\widetilde{m}_i)_{i\in K} \in \{0,1\}^{|K|}$.*

– **Decomposable Correctness.** *For every $\lambda \in \mathbb{N}^\star$, $K \subseteq [k']$, every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, and every $\vec{m} = ((m_0^{(i)}, m_1^{(i)}))_{i\in[k']} \in \{0,1\}^{2k'}$,*

$$\Pr\left[(\widetilde{m}_1, \ldots, \widetilde{m}_{|K|}) = (m_{\sigma_i}^{(i)})_{i\in K} : \begin{array}{r} (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \\ ((\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}}), \mathsf{rep}) \xleftarrow{\$} \mathsf{rep\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}, \mathsf{rep}) \\ (\widetilde{m}_1, \ldots, \widetilde{m}_{|K|}) \xleftarrow{\$} \mathsf{rep\text{-}OTD}(K, (\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K), \mathsf{rep}, \mathsf{st}) \end{array} \right] = 1 ,$$

$$\textit{where } \sigma_i \coloneqq b_{\max\{j \colon (\sum_{j'<j} \mathsf{rep}_{j'}) \le i\}} .$$

– **Receiver Security (against Semi-Honest Sender).** *There exists an expected polynomial time simulator* Sim_S *such that for every $\lambda \in \mathbb{N}^\star$ and every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$,*

$$\left\{ \mathsf{otr} \colon (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \right\} \overset{c}{\approx} \left\{ \mathsf{Sim}_S(1^\lambda) \right\} .$$

– **Decomposable Sender Security (against Semi-Honest Receiver).** *There exists an expected polynomial time simulator* Sim_R *such that for every $\lambda \in \mathbb{N}^\star$, every* $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0,T]^k$ *such that* $\|\mathsf{rep}\|_1 = k'$*, every $K \subseteq [k']$, every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, and every $\vec{m} =$*

$((m_0^{(i)}, m_1^{(i)}))_{i\in[k']} \in \{0,1\}^{2k'}$,

$$\left\{(\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K, \mathsf{otr}, \mathsf{st}) : \begin{array}{c}(\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{rep\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}, \mathsf{rep})\end{array}\right\} \overset{\mathrm{c}}{\approx}$$

$$\left\{(\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}, \mathsf{otr}, \mathsf{st}) : \begin{array}{c}(\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{Sim}_R(1^\lambda, K, (m_{\sigma_i}^{(i)})_{i\in K}, \vec{b}, \mathsf{rep}, \mathsf{otr}, \mathsf{st})\end{array}\right\}$$

$$\textit{where } \sigma_i := b_{\max\{j\colon (\sum_{j'<j} \mathsf{rep}_{j'}) \le i\}} \ .$$

Decomposable Two-Round Batch Oblivious Transfer with Bounded Query Repetition

Parameters: Batch number k, Repetition bound T.
Requires: A two-round decomposable batch dec-OT protocol $\mathsf{dec\text{-}OT} = (\mathsf{dec\text{-}OTR}, \mathsf{dec\text{-}OTS}, \mathsf{dec\text{-}OTD})$ with α-overhead such that $\alpha(k) = o(k/T)$.

rep-OTR**:** On input the security parameter 1^λ and a vector of selection bits $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$:

```
1. Compute $(\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b})$.
2. Output $(\mathsf{otr}, \mathsf{st})$.
```

rep-OTS**:** On input the security parameter 1^λ, a receiver message otr, a database $\vec{m} \in \{0,1\}^{2k'}$, and a vector of repetitions $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0, T]^k$ such that $\|\mathsf{rep}\|_1 = k'$:

```
1. Parse $\vec{m}$ as $((m_0^{(j,i)}, m_1^{(j,i)}))_{j\in[k], i\in[\mathsf{rep}_j]}$ .
   // The first $\mathsf{rep}_1$ pairs are indexed by $j = 1$, the next $\mathsf{rep}_2$ by $j = 2$, and so on.
2. For $j \in [k]$ and $i \in [\mathsf{rep}_j + 1, T]$, set $(m_0^{(j,i)}, m_1^{(j,i)}) \leftarrow (0, 0)$.
   // The database is padded with "dummy" elements so there are exactly $T$ pairs associated with each $j \in [k]$.
3. For $i \in [T]$ set $\vec{m}_i := ((m_0^{(j,i)}, m_1^{(j,i)}))_{j\in[k]}$.
   // Each "sub-database" $\vec{m}_i$ contains the $i^{\text{th}}$ pair associated with each $j \in [k]$.
4. For $j = 1 \ldots T$:
       Compute $(\mathsf{ot}^\star_{S,j}, \mathsf{ot}^{\mathsf{dec}}_{S,j}) \xleftarrow{\$} \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}_j)$.
5. Set $\mathsf{ots}^\star \leftarrow \mathsf{ot}^\star_{S,1} \| \ldots \| \mathsf{ot}^\star_{S,T}$ and $\mathsf{ot}^{\mathsf{dec}}_S \leftarrow ([\mathsf{ot}^{\mathsf{dec}}_{S,1}]_1 \| \ldots \| [\mathsf{ot}^{\mathsf{dec}}_{S,\mathsf{rep}_1}]_1) \| \ldots \| ([\mathsf{ot}^{\mathsf{dec}}_{S,1}]_k \| \ldots \| [\mathsf{ot}^{\mathsf{dec}}_{S,\mathsf{rep}_k}]_k)$.
   // Filter out the bits of the decomposable parts associated with dummy elements, and reorder the remaining bits according to the original database.
6. Output $(\mathsf{ot}^\star_S, \mathsf{ot}^{\mathsf{dec}}_S)$.
```

rep-OTD**:** On input a sender message ots, a vector of repetitions $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0, T]^k$ such that $\|\mathsf{rep}\|_1 = k'$, and an internal state st:

```
1. Parse ots as $(\mathsf{ot}^\star_S, \mathsf{ot}^{\mathsf{dec}}_S)$.
2. Parse $\mathsf{ot}^\star_S$ as $\mathsf{ot}^\star_{S,1} \| \ldots \| \mathsf{ot}^\star_{S,T}$.
3. Parse $\mathsf{ot}^{\mathsf{dec}}_S$ as $([\mathsf{ot}^{\mathsf{dec}}_{S,1}]_1 \| \ldots \| [\mathsf{ot}^{\mathsf{dec}}_{S,\mathsf{rep}_1}]_1) \| \ldots \| ([\mathsf{ot}^{\mathsf{dec}}_{S,1}]_k \| \ldots \| [\mathsf{ot}^{\mathsf{dec}}_{S,\mathsf{rep}_k}]_k)$.
4. For $i = 1 \ldots T$:
   (a) Set $K_i := \{j\colon j \in [k], \mathsf{rep}_j \le i\}$, ordered according to the natural order on $\mathbb{N}$.
   (b) Set $\vec{v}_i \leftarrow \|_{j\in K_i} [\mathsf{ot}^{\mathsf{dec}}_{S,i}]_j$.
   (c) Compute $(\widetilde{m}_{i,j})_{j\in K_i} \xleftarrow{\$} \mathsf{dec\text{-}OTD}(K_i, (\mathsf{ot}^\star_{S,i}, v_i), \mathsf{st})$.
       //Note that the size-$|K_j|$ vector output by dec-OTD is indexed here by elements of $K_i$, not by $1, \ldots, |K_i|$.
5. Output $(\widetilde{m}_{1,1}, \ldots, \widetilde{m}_{\mathsf{rep}_1,1}, \widetilde{m}_{1,2}, \ldots, \widetilde{m}_{\mathsf{rep}_2,2}, \ldots, \widetilde{m}_{\mathsf{rep}_k,k})$.
```

Fig. 3: From dec-OT with α overhead to rep-OT with $\alpha \cdot T$ overhead.

Lemma 11 (From $\mathsf{dec\text{-}OT}$ to $\mathsf{rep\text{-}OT}$). *If* $\mathsf{dec\text{-}OT}$ *is a semi-honest two-round decomposable batch OT protocol with* α *overhead, then the construction* $\mathsf{rep\text{-}OT}$ *from fig. 3 is a semi-honest two-round decomposable batch OT protocol with* $\alpha \cdot T$ *overhead and* T*-bounded repetitions.*

Proof.

- *Syntax, Size, and Correctness:* The fact that $\mathsf{rep\text{-}OT}$ fulfills the syntactic requirements as well as correctness follows from inspection. In particular, note that the receiver message output by $\mathsf{rep\text{-}OTR}$ has indeed size $\mathcal{O}(k)$ and that the sender message output by $\mathsf{rep\text{-}OTS}$ is indeed comprised of a reusable part of size $T \cdot \alpha(k)$ and a decomposable part of size k'.

- *Receiver Security:* By receiver security of $\mathsf{dec\text{-}OT}$ there exists an expected polynomial time simulator Sim_S such that for every $\lambda \in \mathbb{N}^\star$ and every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$,

$$\left\{\mathsf{otr} \colon (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b})\right\} \overset{c}{\approx} \mathsf{Sim}_S(1^\lambda) \; .$$

By construction $\mathsf{rep\text{-}OTR} = \mathsf{dec\text{-}OTR}$, so we also have that

$$\left\{\mathsf{otr} \colon (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b})\right\} \overset{c}{\approx} \mathsf{Sim}_S(1^\lambda) \; ,$$

which concludes this part of the proof.

- *Sender Security:* By sender security of $\mathsf{dec\text{-}OT}$ there exists an expected polynomial time simulator $\mathsf{Sim}_R^{\mathsf{dec}}$ such that for every $\lambda \in \mathbb{N}^\star$, every $K \subseteq [k]$, every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, and every $\vec{m} = ((m_0^{(i)}, m_1^{(i)}))_{i \in [k]} \in \{0,1\}^{2k}$,

$$\left\{(\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K, \mathsf{otr}, \mathsf{st}) \colon \begin{array}{c} (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{BD}}) \xleftarrow{\$} \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}) \end{array}\right\} \overset{c}{\approx}$$
$$\left\{(\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}, \mathsf{otr}, \mathsf{st}) \colon \begin{array}{c} (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{Sim}_R^{\mathsf{dec}}(1^\lambda, K, (m_{b_i}^{(i)})_{i \in K}, \vec{b}, \mathsf{otr}, \mathsf{st}) \end{array}\right\} . \quad (1)$$

The construction of $\mathsf{rep\text{-}OTS}$ makes T parallel calls to $\mathsf{dec\text{-}OTS}$. Sender security follows from a straightforward hybrid argument, replacing these calls one by one with $\mathsf{Sim}_R^{\mathsf{dec}}$; indistinguishability of these hybrids follows by invoking eq. (1) once at each step (and therefore a polynomial number overall).

In a bit more detail, let $\mathsf{rep} = (\mathsf{rep}_1, \ldots, \mathsf{rep}_k) \in [0, T]^k$ such that $\sum_{i=1}^k \mathsf{rep}_i = k'$, and consider the following family of simulators $(\mathsf{Sim}_{R,t}^{\mathsf{rep}})_{t \in [0,T]}$ (fig. 4).

Simulator $\mathsf{Sim}_{R,t}^{\mathsf{rep}}$

Parameters: For $j \in [k]$, $\mathsf{ind}(j) := \sum_{j' < j} \mathsf{rep}_{j'}$. We define $\mathsf{First}_t := \bigcup_{j=1}^k [\mathsf{ind}(j),\ \mathsf{ind}(j) + \min(\mathsf{rep}_j, t-1)]$. For $i \in [k']$, $\sigma_i := b_{\max\{j \colon (\sum_{j'<j} \mathsf{rep}_{j'}) \leq i\}}$.

On input $(1^\lambda, K \subseteq [k'], (m_{\sigma_i}^{(i)})_{i \in K \cap \mathsf{First}_t}, (m_0^{(i)}, m_1^{(i)})_{i \in [k'] \setminus \mathsf{First}_t}, \mathsf{otr}, \mathsf{st})$ where :
// The datase is comprised of rep_j pairs "to be selected according to b_j", for each $j \in [k]$. For each j, the simulator is only given the selected message from the first $\min(\mathsf{rep}_j, t)$ pairs, but is given both messages (a "complete pair") from the other $\min(\mathsf{rep}_j, t) - t$ pairs.

1. Parse the family $(m_{\sigma_i}^{(i)})_{i \in K \cap \mathsf{First}_t}$ as $(m_{\sigma_{j,i}}^{(j,i)})_{(j,i) \in K'}$ where K' is the subset of $\{(j,i) \colon j \in [k], i \in [\mathsf{rep}_j]\}$ defined by $(j,i) \in K' \Leftrightarrow ((\sum_{j'<j} \mathsf{rep}_j) + i) \in K \cap \mathsf{First}_t$.
   // Each $i \in [k']$ can uniquely be associated with a pair (j,i) where $j \in [k]$ and $i \in [\mathsf{rep}_j]$. The simulator computes this re-indexing for the database elements for which it does not have a complete pair...

```
2. Parse the family $(m_0^{(i)}, m_1^{(i)})_{i\in[k']\setminus\mathsf{First}_t}$ as $((m_0^{(j,i)}, m_1^{(j,i)}))_{j\in[k],i\in[t+1,rep_j]}$ where $m_b^{(j,i)} := m_b^{(\mathsf{ind}(j)+i)}$.
   // ... and for those where he is given a complete pair.
3. Complete the family $(m_{\sigma_{j,i}}^{(j,i)})_{(j,i)\in K'}$ as $((m_0^{(j,i)}, m_1^{(j,i)}))_{j\in[k],i\in[1,t]}$, where $m_{1-\sigma_{j,i}}^{(j,i)} := 0$ for $(j,i)\in K'$ and $(m_0^{(i)}, m_1^{(i)}) := (0,0)$ for $(j,i)\notin K'$.
4. For $j\in[k]$ and $i\in[\mathsf{rep}_j+1,T]$, set $(m_0^{(j,i)}, m_1^{(j,i)}) \leftarrow (0,0)$.
   // This "padding" of the database is also performed in the real execution and is not due to the simulator not being given the entire database.
5. For $i\in[T]$ set $\vec{m}_i := ((m_0^{(j,i)}, m_1^{(j,i)}))_{j\in[k]}$.
6. For $i = 1\dots t$:
       Compute $(\mathsf{sim}_i^\star, \mathsf{sim}_i^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{Sim}_R^{\mathsf{dec}}(1^\lambda, [T], \vec{m}_i, \vec{b}, \mathsf{otr}, \mathsf{st})$
7. For $i = (t+1)\dots T$:
       Compute $(\mathsf{ot}_{S,i}^\star, \mathsf{ot}_{S,i}^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}_i)$.
8. Set $\mathsf{ots}^\star \leftarrow \mathsf{sim}_1^\star \| \dots \| \mathsf{sim}_t^\star \| \mathsf{ot}_{S,t+1}^\star \| \dots \| \mathsf{ot}_{S,T}^\star$ .
9. Set $\mathsf{ot}_S^{\mathsf{dec}} \leftarrow \|_{j=1}^{k} \left([\mathsf{sim}_1^{\mathsf{dec}}]_j \| \dots \| [\mathsf{sim}_{\min(t,\mathsf{rep}_j)}^{\mathsf{dec}}]_j \| [\mathsf{ot}_{S,\min(t,\mathsf{rep}_j)+1}^{\mathsf{dec}}]_j \| \dots \| [\mathsf{ot}_{S,\mathsf{rep}_j}^{\mathsf{dec}}]_j\right)$.
10. Output $(\mathsf{ot}_S^\star, [\mathsf{ot}_S^{\mathsf{dec}}]_K)$.
```

Fig. 4: Simulator $\mathsf{Sim}_{R,t}^{\mathsf{rep}}$ replacing the first t calls to $\mathsf{dec\text{-}OTS}$ by calls to $\mathsf{Sim}_S^{\mathsf{dec}}$.

Consider the following distributions:

$$\Delta^{\mathsf{real}} := \left\{ (\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K, \mathsf{otr}, \mathsf{st}) : \begin{array}{c} (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{rep\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}, \mathsf{rep}) \end{array} \right\}$$

$$\forall t\in[T],\ \Delta_t^{\mathsf{Sim}} := \left\{ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}, \mathsf{otr}, \mathsf{st}) : \begin{array}{r} (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{Sim}_{R,t}^{\mathsf{rep}}(1^\lambda, K, (m_{\sigma_i}^{(i)})_{i\in K\cap\mathsf{First}_t}, \\ (m_0^{(i)}, m_1^{(i)})_{i\in[k']\setminus\mathsf{First}_t}, \vec{b}, \mathsf{rep}, \mathsf{otr}, \mathsf{st}) \end{array} \right\}$$

where $\sigma_i := b_{\max\{j : (\sum_{j'<j} \mathsf{rep}_{j'}) \leq i\}}$.

For all $t\in[T]$, it holds that $\Delta_{t-1}^{\mathsf{Sim}} \overset{c}{\approx} \Delta_t^{\mathsf{Sim}}$ by decomposable sender security of $\mathsf{dec\text{-}OT}$. Indeed, if a PPT $\mathcal{D}$ could distinguish $\Delta_{t-1}^{\mathsf{Sim}} \overset{c}{\approx} \Delta_t^{\mathsf{Sim}}$ with probability ϵ then it could distinguish the following distributions with the same probability:

$$\left\{ (\mathsf{ots}^\star, [\mathsf{ots}^{\mathsf{dec}}]_K, \mathsf{otr}, \mathsf{st}) : \begin{array}{c} (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{BD}}) \xleftarrow{\$} \mathsf{dec\text{-}OTS}(1^\lambda, \mathsf{otr}, \vec{m}_t) \end{array} \right\}$$

where $\vec{m}_t$ is defined in line 6. of $\mathsf{Sim}_{R,t}^{\mathsf{rep}}$

and

$$\left\{ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}, \mathsf{otr}, \mathsf{st}) : \begin{array}{c} (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{dec\text{-}OTR}(1^\lambda, \vec{b}) \\ (\mathsf{sim}^\star, \mathsf{sim}^{\mathsf{dec}}) \xleftarrow{\$} \mathsf{Sim}_R^{\mathsf{dec}}(1^\lambda, K, (m_{\sigma_i}^{(i)})_{i\in K\cap[\mathsf{ind}(j), \mathsf{ind}(j)+\min(t-1,\mathsf{rep}_j)]}, \vec{b}, \mathsf{otr}, \mathsf{st}) \end{array} \right\}.$$

But by eq. (1), they are computationally indistinguishable and therefore ϵ must be negligible.

Observe that $\Delta^{\mathsf{real}} = \Delta_0^{\mathsf{Sim}} \overset{c}{\approx} \dots \overset{c}{\approx} \Delta_T^{\mathsf{Sim}}$. Since $[k']\setminus\mathsf{First}_T = \varnothing$, $\mathsf{Sim}_{R,T}^{\mathsf{rep}}$ only takes as input $(1^\lambda, K, (m_{\sigma_i}^{(i)})_{i\in K}, \vec{b}, \mathsf{rep}, \mathsf{otr}, \mathsf{st})$ and therefore we have shown our construction has decomposable sender security.

□

4.4 Two-Round Batch SPIR with Correlated Queries from Two-Round Decomposable Batch OT (with Bounded Query Repetitions)

In this section, definition 12 defines *"mix-and-match functions"* and lemma 13 shows how they can be built. Definition 14 introduces the notion of *two-round batch SPIR protocol with correlated "mix and match" queries* (corrSPIR), and theorem 15 provides a reduction from rep-OT to corrSPIR, with an explicit transformation in fig. 7.

We next introduce and achieve a notion of batch symmetric PIR with correlated queries. This corresponds to batch SPIR where the queries are not independent; rather, the total entropy w used to describe the queries is small, and the queried indices can be reconstructed via a public function that "mixes and matches" the individual bits of entropy in a public manner. In more detail, if the w bits of entropy are $\alpha_1, \ldots, \alpha_w$, "mixing and matching" means that each of the $(n = \log N)$-bit queries to a single database can be obtained by concatenating n of the bits α_i, possibly permuted. In the notation below, the j^{th} query is given by vector $(\alpha_{s_{j,1}}, \ldots, \alpha_{s_{j,n}})$ (in other words, the j^{th} query is associated with the ordered subset $S_j = \{s_{j,1}, \ldots, s_{j,n}\}$ of the bits of entropy). This notion is tailor-made for our application to sublinear computation, but may be of independent interest. Let us now sketch the construction, and highlight both the need for decomposability and why we need the queries to be correlated.

Our starting point is the observation originally present in [KO97] (and later re-used explicitly in [IP07, DGI+19]) using the following Merkle tree abstraction that a rate-1 two-round 1-out-of-2 string OT can be seen as a hash function with a compression factor of two, and can be used to build (block) symmetric PIR. Let us sketch the construction under the (idealised) assumption that we have access to a rate-1 two-round 1-out-of-2 bit OT primitive, which is better suited to our purposes than its string variant. Suppose the server holds a database of $N = 2^n$ bits and that the client wants to retrieve the element stored at index $x = (x_1, \ldots, x_n)$. If the client sends a receiver message $\mathsf{otr}_1 \leftarrow \mathsf{OTR}(x_1)$ for the first bit of the desired index, the server can take the database, pair up elements whose indices differ only on the first bit, then apply the "hash function" $\mathsf{OTS}(\mathsf{otr}_1, \cdot)$ in order to retrieve a single-bit "hashed value" for each pair. If the server were to send all $N/2$ "hashes", the client could retrieve exactly the elements of the database whose indices start with x_1 by applying $\mathsf{OTD}(\mathsf{st}_1, \cdot)$ (st_1 was generated alongside otr_1). If instead the client sends receiver messages $(\mathsf{otr}_1, \ldots, \mathsf{otr}_n)$, one for each bit of the desired index, the server can now iteratively compress the database down to a single bit by building a "Merkle tree" using $\mathsf{OTS}(\mathsf{otr}_d, \cdot)$ at every node of depth $n - d$. If the server sends this single-bit root of the tree, the client can retrieve the element at index $x_1 \ldots x_n$ by iteratively applying $\mathsf{OTD}(\mathsf{st}_d, \cdot)$ for $d = n, \ldots, 1$. In fact, this is the only element of the database which the client can recover from the root of the Merkle tree (intuitively, when the server applied $\mathsf{OTS}(\mathsf{otr}_d, \cdot)$ they removed the client's ability to retrieve any information about elements whose indices have $1 - x_d$ in position d).

The above construction achieves SPIR with optimal communication, from the idealised primitive of rate-1 two-round 1-out-of-2 bit OT. We may ask whether we can replace this primitive with a more realistic batched version, and have the client send $\mathsf{BatchOTR}(x_1, \ldots, x_n)$ for instance in the hopes the client can batch the OT sender messages it has to compute. Unfortunately, while the server has to compute N OT sender messages with a first selection bit, then $N/2$ OT sender messages with a second, and so on, the messages at each layer are crafted adaptively and therefore cannot be batched.

Now consider the setting where the server holds a batch of k databases. If the sender is to compress each database down to a single bit using the "Merkle tree" approach, it has to compute $N/2^d$ OT sender messages for each layer of $d = 1, \ldots, n$ of each of the k Merkle trees. While messages across layers cannot be batched, OTs from the same layer of different trees can! The main challenge is that we can only afford (in order to keep communication low) to use a single batched receiver message in order to compute all of the sender messages. This requires a special assumption on the queries, which need to be highly correlated for this approach to work. We will be interested in how many times a given α_i appears within the k queries (counted by the occurrence function t_i below), as well as how many times it appears in specific position $j' \in [n]$ within the k queries (denoted below by $t_{i,j'}$). If all $t_{i,j'}$ are bounded by T, then for each level $j' \in [n]$ in the "Merkle forest" we can achieve the desired length-halving compression by using at most T batch OT sender computations on the original batch OT selection vector $\vec{\alpha}$.

Definition 12 ("Mix and Match" Functions). *A "mix and match" function* $\mathsf{MixAndMatch}\colon \{0,1\}^w \to [N]^k$ *is one parameterised by* k *ordered subsets of* $n := \log N$ *elements of* $[w]$, $S_j = (s_{j,1}, \dots, s_{j,n}) \in [w]^n$ *for* $j \in [k]$ *such that:*

$$\forall \vec{\alpha} = (\alpha_1, \dots, \alpha_w) \in \{0,1\}^w, \mathsf{MixAndMatch}(\alpha_1, \dots, \alpha_w) := (x_1, \dots, x_k),$$
$$\text{with } x_j := \alpha_{s_{j,1}} \cdots \alpha_{s_{j,n}} \in [N].$$

Such a function is associated with an occurrence function, which counts the occurrences of each input position in the outputs:

$$\begin{aligned} t_\cdot\colon\ & [w] \to [k] \\ & i \ \mapsto t_i = \sum_{j=1}^{k} \mathbf{1}_{i \in S_j} \end{aligned}$$

Each t_i $(i \in [w])$ *can be decomposed as* $t_i = t_{i,1} + \dots + t_{i,n}$, *where* $t_{i,j'}$ *is equal to the number of values of* $j \in [k]$ *such that* $s_{j,j'} = i$.

- $\mathsf{MixAndMatch}$ *is said to be* T-balanced *if* $\forall i \in [w], \forall j' \in [n], t_{i,j'} \le T$.
- $\mathsf{MixAndMatch}$ *is said to be* T-balanceable *if it can be expressed as the function* $\mathsf{MixAndMatch} = (\mathsf{MixAndMatch}' \circ \mathsf{replicate})$, *where* $\mathsf{MixAndMatch}\colon \{0,1\}^{w'} \to [N]^k$ *is a* T*-balanced mix-and-match function and* $\mathsf{replicate}$ *is defined as:*

$$\begin{aligned} \mathsf{replicate}\colon\quad & \{0,1\}^w \quad \to \{0,1\}^{w'} \\ & (b_1, \dots, b_w) \mapsto (b_1^{\| \lceil t_1/T \rceil} \| \dots \| b_w^{\| \lceil t_w/T \rceil}) \end{aligned} \qquad \text{where } w' := \sum_{i \in [w]} \lceil t_i/T \rceil.$$

Lemma 13. *Let* $w, n \in \mathbb{N}$ *be a sufficiently large integers. For any family of unordered subsets* $S_1, \dots, S_k \in \binom{[w]}{n}$ *there exists an ordering of each subset* S_j *such that the mix-and-match function induced by the resulting* $(\tilde{S}_j)_{j \in [k]}$ *is* $\mathsf{polylog}(w)$*-balanceable.*
Furthermore, such orderings can be found in expected constant time.

Proof. Let us prove, via a Chernoff bound, that reordering (after replication) each subset independently and uniformly at random, yields a $\mathsf{polylog}(w)$-balanced mix-and-match function with high probability. It follows that any family of subsets characterises a T balanceable mix-and-match function.

Preliminary Notations. Let $S_1 = \{s_{1,1}, \dots, s_{1,n}\}, \dots, S_k = \{s_{k,1}, \dots, s_{k,n}\} \in \binom{[w]}{n}$ be a family of unordered sets, and consider the associated occurrence function, which counts the occurrences of each input position in each subset:

$$\begin{aligned} t_\cdot\colon\ & [w] \to [k] \\ & i \ \mapsto t_i = \sum_{j=1}^{k} \mathbf{1}_{i \in S_j} \end{aligned}$$

Define $w' := \sum_{i \in [w]} \lceil t_i/T \rceil$.

Analysis of Randomized Construction. Consider the i.i.d. random variables $\pi_1, \dots, \pi_k \hookleftarrow \mathcal{U}(\mathfrak{S}_{w'})$ (where $\mathcal{U}(\mathfrak{S}_{[w']})$ is the uniform distribution on all permutations of $[w']$). Define the random variables $\tilde{S}_1, \dots \tilde{S}_k$ as the following deterministic functions of the random variables $\pi_1, \dots, \pi_k$: for each $j \in [k]$, $\tilde{S}_j := (s_{j,\pi_j(1)}, \dots, s_{j,\pi_j(w')})$. Finally, define the indicator random variables $(t_{i,j,d})_{i \in [w'], j \in [k], d \in [n]}$ as the following deterministic functions of the $(\pi_j)_{j \in [k]}$: for each $i \in [w'], j \in [k], d \in [n]$, $t_{i,j,d} := \mathbf{1}_{i == s_{j,d}}$.

Observe that the event "$\tilde{S}_1, \dots \tilde{S}_k$ characterizes a T-balanced mix-and-match function" (for any T) is equivalent to the event "$\forall d \in [n], \forall i \in [w'], \sum_{j \in k} t_{i,j,d} \le T$". Since the $(\pi_j)_{j \in [k]}$ are independent, so are the $(t_{i,j,d})_{j \in [k], i \in [w']}$ for any fixed $d \in [k]$. Further note that $\forall d \in [k], \mu_d := \mathbb{E}(\sum_{j \in k, i \in [w']} t_{i,j,d}) = \sum_{j \in k, i \in [w']} \mathbb{E}(t_{i,j,d}) = k \cdot \frac{\sum_{i \in [w]} t_i}{d}$. Therefore, by a Chernoff bound[9], for every $d \in [n]$, $\Pr(\sum_{j \in k} t_{i,j,d} > T) < 1/\lambda^{\omega(1)}$. By union-bound, $\Pr[\forall d \in [n], \forall i \in [w'], \sum_{j \in k} t_{i,j,d} \le T] \le n \cdot w'/\lambda^{\omega(1)} = 1/\lambda^{\omega(1)}$. $\square$

[9] Specifically, we are using the Chernoff bound in the form which is standardly denoted "$\Pr[X > (1+\delta)\mu] < \exp(-\delta^2 \mu(2+\delta))$".

Definition 14 (Two-Round Batch Computational Batch SPIR with Correlated "Mix and Match" Queries). *A semi-honest two-round batch SPIR protocol with correlated "mix and match" queries between a sender and a receiver can be defined as a triple of PPT algorithms* $\mathsf{corrSPIR} = (\mathsf{corrSPIR}_R, \mathsf{corrSPIR}_S, \mathsf{corrSPIR}_D)$ *parameterised by a public* T-balanceable *"mix and match" function (definition 12)* $\mathsf{MixAndMatch}\colon \{0,1\}^w \to [N]^k$ *with the following syntax and properties:*

- **Syntax.**
  - $\mathsf{corrSPIR}_R$: *On input the security parameter* 1^λ *and a vector of selection bits* $\vec{b} = (b_1, \ldots, b_w) \in \{0,1\}^w$, $\mathsf{corrSPIR}_R$ *outputs a receiver message* $\mathsf{spir}_R \in \{0,1\}^{\mathcal{O}(w)}$ *and an internal state* st*; without loss of generality, we assume* st *contains all the coins used by* $\mathsf{corrSPIR}_R$ *as well as* $\vec{b}$.
  - $\mathsf{corrSPIR}_S$: *On input the security parameter* 1^λ*, a receiver message* spir_R*, and k N-bit databases* $\vec{m}_1, \ldots, \vec{m}_k \in \{0,1\}^N$, $\mathsf{corrSPIR}_S$ *outputs a sender message* $\mathsf{spir}_S \in \{0,1\}^{\mathcal{O}(k)}$.
  - $\mathsf{corrSPIR}_D$: *On input a sender message* spir_S *and an internal state* st, $\mathsf{corrSPIR}_D$ *outputs a vector of messages* $(\widetilde{m}_1, \ldots, \widetilde{m}_k) \in \{0,1\}^k$.
- **Correctness.**

$$\forall \vec{b} = (b_1, \ldots, b_w) \in \{0,1\}^w, \forall \vec{M} = (\vec{m}_1, \ldots, \vec{m}_k) \in \{0,1\}^{N\cdot k},$$

$$\Pr\left[(\widetilde{m}_1, \ldots, \widetilde{m}_k) = (\vec{m}_1[x_1], \ldots, \vec{m}_k[x_k]) \colon \begin{array}{r} (\mathsf{spir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \vec{b}) \\ \mathsf{spir}_S \xleftarrow{\$} \mathsf{corrSPIR}_S(1^\lambda, \mathsf{spir}_R, \vec{M}) \\ (\widetilde{m}_1, \ldots, \widetilde{m}_k) \xleftarrow{\$} \mathsf{corrSPIR}_D(\mathsf{spir}_S, \mathsf{st}) \end{array}\right] = 1$$

$$\textit{where } (x_1, \ldots, x_k) \coloneqq \mathsf{MixAndMatch}(\vec{b}).$$

- **Security.** *The following protocol,* Π_{corrSPIR} *(fig. 5), securely realises* $\mathcal{F}_{\mathsf{corrSPIR}}$ *(fig. 6) in the presence of a semi-honest adversary. The receiver computes* $(\mathsf{spir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \vec{b})$ *and sends* spir_R *to the sender, who in turn computes* $\mathsf{spir}_S \xleftarrow{\$} \mathsf{corrSPIR}_S(1^\lambda, \mathsf{spir}_R, \vec{M})$ *and returns* spir_S*; finally, the receiver computes and outputs* $(\widetilde{m}_1, \ldots, \widetilde{m}_k) \xleftarrow{\$} \mathsf{corrSPIR}_D(\mathsf{spir}_S, \mathsf{st})$.

Receiver		Sender
`Input:` $\vec{b} \in \{0,1\}^w$		`Input:` $\vec{M} \in \{0,1\}^{N\cdot k}$
$(\mathsf{spir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \vec{b})$		
	$\xrightarrow{\quad \mathsf{spir}_R \quad}$	
		$\mathsf{spir}_S \xleftarrow{\$} \mathsf{corrSPIR}_S(1^\lambda, \mathsf{spir}_R, \vec{M})$
	$\xleftarrow{\quad \mathsf{spir}_S \quad}$	
$(\widetilde{m}_1, \ldots, \widetilde{m}_k) \xleftarrow{\$} \mathsf{corrSPIR}_D(\mathsf{spir}_S, \mathsf{st})$		
`Output:` $(\widetilde{m}_1, \ldots, \widetilde{m}_k)$		

Fig. 5: Two-Round $\mathsf{corrSPIR}$ Protocol Π_{corrSPIR}.

Functionality $\mathcal{F}_{\mathsf{corrSPIR}}$

The functionality $\mathcal{F}_{\mathsf{corrSPIR}}$ is parameterised by the number k of SPIRs in the batch, the size N of each database, and the number w of selection bits. Furthermore, it is parameterised by a public T-balanceable "mix and match" function (definition 12) $\mathsf{MixAndMatch}\colon \{0,1\}^w \to [N]^k$. $\mathcal{F}_{\mathsf{corrSPIR}}$ interacts with an ideal sender $\mathbf{S}$ and an ideal receiver $\mathbf{R}$ via the following queries.

1. On input $(\texttt{sender}, \vec{M} = (\vec{m}_i)_{i\in[k]})$ from $\mathbf{S}$, with $\vec{m}_i = (m_{i,j})_{j\in[N]} \in \{0,1\}^N$ store $\vec{M}$.

2. On input $(\texttt{receiver}, (b_j)_{j\in[w]})$ from $\mathbf{R}$, check if a tuple of inputs $\vec{M}$ has already been recorded; if so, compute $(x_1, \ldots, x_k) \coloneqq \mathsf{MixAndMatch}(b_1, \ldots, b_w) \in [N]^k$, send $(m_{i,x_i})_{i\in[k]}$ to $\mathbf{R}$, and halt.

If the functionality receives an incorrectly formatted input, it aborts.

Fig. 6: Ideal Functionality $\mathcal{F}_{\mathsf{corrSPIR}}$ for Batch SPIR with Correlated "Mix and Match" Queries

Batch SPIR with Correlated "Mix and Match" Queries

Parameters: k, N, $n \coloneqq \log N$, w, T, a T-balanceable $\mathsf{MixAndMatch}\colon \{0,1\}^w \to [N]^k$ (parameterised by subsets $S_j = (s_{j,1}, \ldots, s_{j,n}) \in [w]^n$ for $j \in [k]$) and an associated list of number of occurrences $(t_1, \ldots, t_w)$ with $t_i = t_{i,1} + \cdots + t_{i,n}$, a two-round batch rep-OT protocol $\mathsf{rep\text{-}OT} = (\mathsf{rep\text{-}OTR}, \mathsf{rep\text{-}OTS}, \mathsf{rep\text{-}OTD})$.

$\mathsf{corrSPIR}_R$**:** On input the security parameter 1^λ and a vector of selection bits $\vec{b} = (b_1, \ldots, b_w) \in \{0,1\}^w$:

```
1. Set $\vec{b}' \leftarrow b_1^{\|\lceil t_1/T\rceil} \| \ldots \| b_w^{\|\lceil t_w/T\rceil}$ .
   // $\vec{b}'$ is the vector whose first $\lceil t_1/T\rceil$ coordinates are equal to $b_1$, followed by $\lceil t_2/T\rceil$ coordinates equal to $b_2$, and so on. Note that the total size of this "selection vector with redundancies" is $\sum_{i=1}^{w} \lceil t_i/T\rceil$.
2. Compute $(\mathsf{spir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{OTR}(1^\lambda, \vec{b}')$, and output $(\mathsf{spir}_R, \mathsf{st}\|\vec{b})$.
```

$\mathsf{corrSPIR}_S$**:** On input the security parameter 1^λ, a receiver message spir_R, and k databases $\vec{m}_1, \ldots, \vec{m}_k \in [N]$:

```
1. Set $(\mathtt{DB}_{1,1}, \ldots, \mathtt{DB}_{1,k}) \coloneqq (\vec{m}_1, \ldots, \vec{m}_k)$.
   // Throughout, $\mathtt{DB}_{d,k}$ will correspond to the values of the $d^{\text{th}}$ layer of the $k^{\text{th}}$ Merkle tree.
2. For $d = 1, \ldots, n$:
   (a) For $i = 1, \ldots, w$:
         Set $\mathsf{rep}_{d,i} \leftarrow \begin{cases} T^{\|\lceil t_{i,d}/T\rceil} \| 0^{\|\lceil t_i/T\rceil - \lceil t_{i,d}/T\rceil} & \text{if } T | t_{i,d} \\ T^{\|\lfloor t_{i,d}/T\rfloor} \| t_{i,d}\%T \| 0^{\|\lceil t_i/T\rceil - \lceil t_{i,d}/T\rceil} & \text{if } T \nmid t_{i,d} \end{cases}$
   (b) Set $\mathsf{rep}_d \leftarrow \mathsf{rep}_{d,1} \| \ldots \| \mathsf{rep}_{d,w}$ .
       // Note that $\mathsf{rep}_d$ is a vector of size $\sum_{i=1}^{w} \lceil t_i/T\rceil$ with elements in $[0,T]$, and such that $\|\mathsf{rep}_d\|_1 = \sum_{i=1}^{w} t_{i,d}$ .
   (c) Initialise $X_d \leftarrow \varnothing$ .
   (d) For $j = 1, \ldots, k$:
         For $x = 0, \ldots, N/2^d - 1$:
            $X_d.\mathsf{append}(((\mathtt{DB}_{d,j}[2x], \mathtt{DB}_{d,j}[2x+1]),\ s_{j,d},\ x,\ j))$ .
            // Note that $X_d$ now contains $k \cdot N/2^d$ elements. For each $i \in [w]$, exactly $t_{i,d} \cdot N/2^d$ of the form $((\cdot,\cdot),\cdot,i,\cdot)$. Indeed, by definition, $t_{i,d} = |\{j \in [k] \colon s_{j,d} = i\}|$.
   (e) Sort $X_d$ according to the lexicographic order which first sorts by increasing fourth element (the "$j \in [k]$") and then, in case of equality, by increasing third element (the "$x \in [0, N/2^d - 1]$").
   (f) Greedily partition $X_d$ as $X_d = X_{d,1} \sqcup \cdots \sqcup X_{d,(N/2^d)}$ such that for each $\ell \in [N/2^d]$ and each $i \in [w]$, $X_{d,\ell}$ contains (up to) $t_{i,d}$ elements of the form $((\cdot,\cdot), i, \cdot, \cdot)$; "greedily" is here taken to mean that the first $t_{i,d}$ elements of the form $((\cdot,\cdot), i, \cdot, \cdot)$ are placed in $X_{d,1}$, the next $t_{i,d}$ in $X_{d,2}$, and so on.
       // Note that each $X_{d,\ell}$ can contain up to $t_{i,d}$ elements of the form $((\cdot,\cdot), i, \cdot, \cdot)$, of which there are a total of $(N/2^d) \cdot t_{i,d}$. Therefore $X_d$ can indeed be decomposed into $(N/2^d)$ such partitions.
       // Further note that each $X_{d,\ell}$ $(\ell \in [N/2^d])$ is a set of size $\sum_{i=1}^{w} t_i'$ .
   (g) For $\ell = 1, \ldots, N/2^d$:
        – Sort $X_{d,\ell}$ according to the second element in increasing order, breaking ties with the
          fourth, and then if necessary the third element of the 4-tuples.
          // After this re-ordering, the first $t_1$ tuples are of the form $((\cdot,\cdot),1,\cdot,\cdot)$, followed by
          $t_2$ tuples of the form $((\cdot,\cdot),2,\cdot,\cdot)$, and so on.
        – Set $\mathtt{DB}'_{d,\ell} \leftarrow (S_{d,\ell}[0].\mathsf{first},\ldots,S_{d,\ell}[(\sum_{i=1}^{w} t_{i,d})-1].\mathsf{first}) \in \{0,1\}^{2|S_{d,\ell}|}$.
          // $\mathtt{DB}'_{d,\ell}$ is obtained by only considering the first of the four entries (which is a pair
          of bits from some $\mathtt{DB}_{d,j}$) of every element of $X_{d,\ell}$.
        – Set $(\mathsf{ots}^\star_{d,\ell},\mathsf{ots}^{\mathsf{dec}}_{d,\ell}) \xleftarrow{\$} \mathsf{rep\text{-}OTS}(1^\lambda,\mathsf{spir}_R,\mathtt{DB}'_{d,\ell},\mathsf{rep}_d)$ .
    (h) If $d < n$:
        – For $j = 1,\ldots,k$:
              Initialise $\mathtt{DB}_{d+1,j} \leftarrow 0^{\|N/2^d}$.
        – For $\ell = 1,\ldots,N/2^d$:
              For $\ell' = 0,\ldots,(\sum_{i=1}^{w} t_{i,d})-1$:
                  Parse $X_{d,\ell}[\ell']$ as $((\cdot,\cdot),\cdot,x,j)$, with $x \in [N/2^d]$ and $j \in [k]$.
                  Set $\mathtt{DB}_{d+1,j}[x] \leftarrow \mathsf{ots}^{\mathsf{dec}}_{d,\ell}[\ell']$ .
    (i) Set $\mathsf{ots}^\star_d \leftarrow (\mathsf{ots}^\star_{d,1},\ldots,\mathsf{ots}^\star_{d,N/2^d})$ .
 3. Set $\mathsf{spir}_S \coloneqq ((\mathsf{ots}^\star_1,\ldots,\mathsf{ots}^\star_n),\mathsf{ots}^{\mathsf{dec}}_n)$, and output $\mathsf{spir}_S$.

$\mathsf{corrSPIR}_D$: On input a sender message $\mathsf{spir}_S$ and an internal state $\mathsf{st}$:

 1. Parse $\mathsf{spir}_S$ as $\mathsf{spir}_S = ((\mathsf{ots}^\star_1,\ldots,\mathsf{ots}^\star_n),\mathsf{ots}^{\mathsf{dec}}_n)$, and parse $\mathsf{st}$ as $\mathsf{st}'\|\vec{b}$ .
 2. Set $(y_1,\ldots,y_k) \leftarrow \mathsf{MixAndMatch}(\vec{b})$ (i.e. $y_j \leftarrow b_{s_{j,1}}\ldots b_{s_{j,n}}$ for $j \in [n]$).
 3. Initialise $(\widetilde{m}_1,\ldots,\widetilde{m}_k) \leftarrow \mathsf{ots}^{\mathsf{dec}}_n$ .
 4. For $d = 1,\ldots,n$ :
    // The goal of this step is to identify which intermediary nodes of the Merkle tree can be
    recovered.
    (a) Initialise $X_d \leftarrow ((\perp,\ s_{j,d},\ x,\ j))_{j\in[k],x\in[0,N/2^d-1]}$
    (b) Sort $X_d$ according to the lexicographic order which first sorts by increasing fourth element
        (the "$j \in [k]$") and then, in case of equality, by increasing third element (the "$x \in
        [0,N/2^d-1]$").
    (c) Greedily partition $X_d$ as $X_d = X_{d,1} \sqcup \cdots \sqcup X_{d,N/2^d}$ such that for each $\ell \in [N/2^d]$ and
        each $i \in [w]$, $X_{d,\ell}$ contains exactly $t_{i,d}$ elements of the form $(\cdot,i,\cdot,\cdot)$; "greedily" is here
        taken to mean that the first $t_{i,d}$ elements of the form $(\cdot,i,\cdot,\cdot)$ are placed in $X_{d,1}$, the
        next $t_{i,d}$ in $X_{d,2}$, and so on.
    (d) For $\ell = 1,\ldots,N/2^d$:
            Sort $X_{d,\ell}$ according to the second element in increasing order, breaking ties with the
            fourth, and then if necessary the third element of the 4-tuples.
    (e) Parse $\mathsf{ots}^\star_d$ as $\mathsf{ots}^\star_d = (\mathsf{ots}^\star_{d,1},\ldots,\mathsf{ots}^\star_{d,N/2^d})$
    (f) For $j = 1,\ldots,k$:
        – Set $\ell_{j,d}$ to be the unique $\ell \in [N/2^d]$ such that $(\perp,\ s_{j,d},\ (b_{s_{j,n}}\ldots b_{s_{j,d}}),\ j) \in X_{d,\ell}$ .
        – Set $\mathsf{ind}_{j,d}$ to be the index of $(\perp,\ s_{j,d},\ (b_{s_{j,n}}\ldots b_{s_{j,d}}),\ j)$ in $X_{d,\ell}$ .
        – Update $\widetilde{m}_j \leftarrow \mathsf{rep\text{-}OTD}(\{\mathsf{ind}_{j,d}\},(\mathsf{ots}^\star_{d,\ell_{j,d}},\widetilde{m}_j),\mathsf{rep},\mathsf{st})$
 5. Output $(\widetilde{m}_1,\ldots,\widetilde{m}_k)$ .
```

Fig. 7: corrSPIR from rep-OT.

Theorem 15. *Assume that* rep-OT *is a semi-honest two-round decomposable batch OT protocol with $\alpha(\cdot)$-overhead and T-bounded query repetitions. Then construction* $(\mathsf{corrSPIR}_R,\mathsf{corrSPIR}_S,\mathsf{corrSPIR}_D)$ *from fig. 7 is a two-round batch SPIR protocol with correlated "mix and match" queries. Furthermore the size of the receiver message is linear in $w + k\cdot n/T$ and the size of the sender message is upper bounded by $k + (\log N)\cdot(N-1)\cdot\alpha(w+k\cdot n/T)$ (where k is the batch number and N is the size of each of the k databases).*

Proof.

– *Size:* The receiver message is a rep-OT receiver message with $\sum_{i=1}^{w}\lceil t_i/T\rceil \le \sum_{i=1}^{w}(1+t_i/T) \le w + (\sum_{i=1}^{w} t_i)/T = w + k\cdot n/T$ selection bits Since rep-OT has upload rate asymptotically one by definition, the receiver message is indeed linear in $w + k \cdot n/T$. The sender message is comprised a single bit per database, as well as $\sum_{d=1}^{n} N/2^d = N - 1$ different "reusable parts" of size $\alpha(w + k \cdot n/T)$. The sender message is therefore of size $k + (\log N) \cdot (N-1) \cdot \alpha(w + k \cdot n/T)$.

– *Correctness:* Correctness mostly follows from inspection, keeping in mind the following description of the instructions (alongside the comments in the pseudocode). Let $d \in [n]$. The pair $(\mathtt{DB}_{d,j}[x'_0], \mathtt{DB}_{d,j}[x'_1])$ corresponds to a pair of elements of $\mathtt{DB}_{d,j}$ whose indices only differ in bit $s_{j,d}$, wish we wish to "hash" down to a single bit using rate-1 OT. Because we only have access to a batched version the OT primitive, the pair will need to be batched with others (in fact, taken from different databases) in such a way that it corresponds to the $s_{j,d}^{\text{th}}$ selection bit. We therefore tag the pair with $s_{j,d}$. Furthermore, we will need to place the "hashed value" (which can be extracted by sender-message decomposability of the batched OT) thus obtained at the correct place in the Merkle tree, *i.e.* in the next level database. For this reason, we additionally tag the pair with (x, j), so as to remember whence it came, and be able to deduce where it should be placed.

– *Security (Standalone Simulation):* We need to show that Π_{corrSPIR} from fig. 5, when instantiated with $\mathsf{corrSPIR} = (\mathsf{corrSPIR}_R, \mathsf{corrSPIR}_S, \mathsf{corrSPIR}_D)$ as defined in fig. 7 securely computes the functionality $f((\vec{m}_i)_{i\in[k]},\ \vec{b}) = (\perp,\ (\vec{m}_i[x_i])_{i\in[k]})$ where $(x_1, \ldots, x_k) := \mathsf{MixAndMatch}(\vec{b})$ in the presence of static semi-honest adversaries.
  - *Corrupted Sender and Honest Receiver.* Because the functionality is deterministic, it suffices to show that there exist a PPT simulator $\mathsf{Sim}_S^{\mathsf{corrSPIR}}$ such that:
$$\{\mathsf{view}_S^{\Pi_{\mathsf{corrSPIR}}}(1^\lambda, (\vec{m}_i)_{i\in[k]}, \vec{b})\} \overset{c}{\approx} \{\mathsf{Sim}_S^{\mathsf{corrSPIR}}(1^\lambda, (\vec{m}_i)_{i\in[k]}, \perp)\}\ .$$
The sender's view in Π_{corrSPIR} consists of its input, internal random tape, and the messages it receives from the receiver. Note that the sender receives a single message, before it sends anything, and therefore its view can be split into two independent parts: the input and coins on one side, and the incoming transcript on the other. Therefore it suffices to show that we can simulate the incoming message given the security parameter and the sender's input. Since rep-OT is secure against a semi-honest sender, by definition there exists an expected polynomial time simulator $\mathsf{Sim}_S^{\mathsf{rep}}$ such that for every $\lambda \in \mathbb{N}^\star$ and every $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$, and if further we define $\vec{b}' \leftarrow b_1^{\|\lceil t_1/T\rceil} \| \ldots \| b_w^{\|\lceil t_w/T\rceil}$,
$$\left\{\mathsf{otr}\colon (\mathsf{otr}, \mathsf{st}) \xleftarrow{\$} \mathsf{rep\text{-}OTR}(1^\lambda, \vec{b}')\right\} \ \overset{c}{\approx}\ \left\{\mathsf{Sim}_S^{\mathsf{rep}}(1^\lambda)\right\}\ .$$
Since the left hand side is exactly the distribution of the unique message received by the sender, it follows that the view of a semi-honest sender can be simulated.

  - *Corrupted Receiver and Honest Sender.* Because the functionality is deterministic, it suffices to show that there exists a PPT simulator $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ such that:
$$\{\mathsf{view}_R^{\Pi_{\mathsf{corrSPIR}}}(1^\lambda, (\vec{m}_j)_{j\in[k]}, \vec{b})\} \overset{c}{\approx} \{\mathsf{Sim}_R^{\mathsf{corrSPIR}}(1^\lambda, \vec{b}, (\vec{m}_j[x_j])_{j\in[k]})\},$$
$$\text{where } (x_1, \ldots, x_k) := \mathsf{MixAndMatch}(\vec{b})\ .$$
Note that the view of the corrupted receiver in Π_{corrSPIR} consists of its input (the vector of selection bits $\vec{b}$), its internal coins, and the single message spir_S it receives from the sender. Consider the simulator $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ which acts as follows:
    1. $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ starts by running the protocol as the receiver would, sampling random coins $\vec{r}_R$ and crafting $\mathsf{spir}_R \leftarrow \mathsf{corrSPIR}_R(1^\lambda, \vec{b}; \vec{r}_R)$;
    2. $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ builds k databases $\vec{m}'_1, \ldots, \vec{m}'_k$ of size N each, containing 0 everywhere except for one position each: the j^{th} database has $\vec{m}[x_j]$ in position x_j (recall that with the knowledge of $\vec{b}$, $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ is able to compute $(x_1, \ldots, x_k) = \mathsf{MixAndMatch}(\vec{b})$);
    3. $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ runs $\mathsf{spir}_S \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \mathsf{spir}_R, (\vec{m}'_j)_{j\in[k]})$.
    4. $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ outputs $(\vec{r}_R, \vec{b}, \mathsf{spir}_S)$.

  In other words, $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ runs in its head an instance of the protocol of fig. 5, but replacing all of the unknown values (*i.e.* all except the $(\vec{m}_j[x_j])_{j\in[k]}$) in the databases $(\vec{m}_j)_{j\in[k]}$ with zeroes.

We will now prove that the view of the corrupted receiver in Π_{corrSPIR} is indistinguishable from the output of the above simulator $\mathsf{Sim}_R^{\mathsf{corrSPIR}}$ via a hybrid argument.

For $j \in [k]$ and $d \in [n]$, let $Y_{j,d} \coloneqq \{x = \overline{x_1 \dots x_n}^{(2)} \in [N] \colon (\forall d' < d, x_{d'} = b_{s_{j,d'}}) \wedge (x_d \neq b_{s_{j,d}})\}$ (in other words, $Y_{j,d}$ is the set of all elements of $[N]$ whose first $d-1$ digits in base two, but not the dth, are the same as those of $\overline{b_{s_{j,1}} \dots b_{s_{j,d}}}^{(2)}$). Observe that for each $j \in [k]$, the $(Y_{j,d})_{d \in [n]}$ form a partition of $[N] \setminus \{\overline{b_{s_{j,1}} \dots b_{s_{j,d}}}^{(2)}\}$. Now, for $d \in [n]$ consider the simulator $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ which acts as follows:

1. $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ starts by running the protocol as the receiver would, sampling random coins $\vec{r}_R$ and crafting $\mathsf{spir}_R \leftarrow \mathsf{corrSPIR}_R(1^\lambda, \vec{b}; \vec{r}_R)$;
2. $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ builds k databases $\vec{m}'_1, \dots, \vec{m}'_k$ of size N each, as follows:

$$\forall j \in [k], \forall x \in [N],\ \vec{m}'_j[x] \coloneqq \begin{cases} \vec{m}_j[x] & \text{if } x \in Y_{j,d} \\ 0 & \text{otherwise} \end{cases}$$

   (recall that with the knowledge of $\vec{b}$, $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ is able to compute $(x_1, \dots, x_k) = \mathsf{MixAndMatch}(\vec{b})$, and therefore also $Y_{j,d}$);
3. $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ runs $\mathsf{spir}_S \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \mathsf{spir}_R, (\vec{m}'_j)_{j \in [k]})$.
4. $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ outputs $(\vec{r}_R, \vec{b}, \mathsf{spir}_S)$.

In other words, $\mathsf{Sim}_{R,d}^{\mathsf{corrSPIR}}$ runs in its head an instance of the protocol of fig. 5, but replacing all of the unknown values (*i.e.* all except the $(\vec{m}_j[x_j])_{j \in Y_{j,d}}$) in the databases $(\vec{m}_j)_{j \in [k]}$ with zeroes.
Observe that for all $j \in [k]$ we have that $Y_{j,1} = [N]$ and $Y_{j,n} = \{x_j\}$, and that $\mathsf{Sim}_{R,1}^{\mathsf{corrSPIR}}$ perfectly simulates the real world, while $\mathsf{Sim}_{R,n}^{\mathsf{corrSPIR}} = \mathsf{Sim}_R^{\mathsf{corrSPIR}}$ lives in the ideal world. For $j \in [1, n-1]$, indistinguishability of the distributions of outputs of $\mathsf{Sim}_{R,j}^{\mathsf{corrSPIR}}$ and $\mathsf{Sim}_{R,j+1}^{\mathsf{corrSPIR}}$ holds by invoking sender security of $\mathsf{rep\text{-}OT}$ k times.

□

4.5 Sublinear Computation of log log-Depth Circuits from corrSPIR

In this section theorem 16 shows how to build sublinear secure computation for shallow (roughly log log-depth) circuits from $\mathsf{corrSPIR}$, with an explicit protocol provided in fig. 8. Main theorem 1 combines all of the previous theorems and shows that sublinear secure computation for shallow circuits can be based on $\mathsf{QR} + \mathsf{LPN}$.

Protocol Π_{2PC}

Functionality:

- **Parameters:** $C \colon \{0,1\}^n \to \{0,1\}^m$ is a boolean circuit of depth k. For $j \in [m]$, $S_j = \{s_{j,1}, \dots, s_{j,2^k}\}$ is the subset[a] of the inputs on which depends the jth output of f, and for $i \in [n]$ we denote t_i the number of outputs of C on which the ith variable depends. $(\pi_j)_{j \in [m]} \in (\mathfrak{S}_{2^k})^m$ is a family of m permutations on $[2^k]$, such that the following is a $(T = \mathsf{polylog}(n))$-balanced "mix and match" function:

$$\begin{array}{rccc} \mathsf{MixAndMatch}_C \colon & \{0,1\}^w & \to & [2^k]^m \\ & (x_1, \dots, x_w) & \mapsto & (x_{s_{j,\pi_j(1)}} \| \dots \| x_{s_{j,\pi_j(2^k)}})_{j \in [m]} \end{array}$$

- **Inputs:** Parties P_0 and P_1 hold additive shares $(\vec{x}_0, \vec{x}_1)$ of an input $\vec{x} \in \{0,1\}^n$.
- **Outputs:** The parties output $C(\vec{x})$.
- **Requires:** $\mathsf{corrSPIR} = (\mathsf{corrSPIR}_R, \mathsf{corrSPIR}_S, \mathsf{corrSPIR}_D)$ is a two-round batch SPIR protocol with correlated "mix and match" queries.

Protocol:

1. P_0 samples $\vec{y}_0 \xleftarrow{\$} \{0,1\}^m$ and for $j \in [m]$ sets $\mathtt{DB}_j \in \{0,1\}^{2^{2^k}}$ to be the truth table of the following function:

$$\begin{array}{rcl} g_j\colon \quad \{0,1\}^{2^k} & \rightarrow & \{0,1\} \\ (X_1,\ldots,X_{2^k}) & \mapsto & C_j((X_{\pi_j(1)} \oplus \vec{x}_0[\pi_j(1)] \| \ldots \| X_{\pi_j(2^k)} \oplus \vec{x}_0[\pi_j(2^k)])) \oplus \vec{y}_0[j] \end{array}$$

where C_j is the j^{th} output of C.

2. P_1 sets $\vec{x}_1' \leftarrow (\vec{x}_1[1])^{\|\lceil t_1'/T\rceil}\| \ldots \| b_w^{\|\lceil t_w'/T\rceil}$.
// $\vec{b}'$ is the vector whose first $\lceil t_1'/T\rceil$ coordinates are equal to b_1, followed by $\lceil t_2'/T\rceil$ coordinates equal to b_2, and so on. Note that the total size of this "selection vector with redundancies" is $\sum_{i=1}^{w}\lceil t_i'/T\rceil$.
3. P_1 samples $(\mathsf{spir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{corrSPIR}_R(1^\lambda, \vec{x}_1)$ and sends spir_R to P_0.
4. P_0 samples $\mathsf{spir}_S \xleftarrow{\$} \mathsf{corrSPIR}_S(1^\lambda, \mathsf{spir}_R, (\mathtt{DB}_j)_{j\in[m]})$ and sends $(\mathsf{spir}_S, \vec{y}_0)$ to P_1.
5. P_1 recovers $\vec{y}_1 \leftarrow \mathsf{corrSPIR}_D(\mathsf{spir}_S, \mathsf{st})$.
6. P_1 sets $\vec{y} \leftarrow \vec{y}_0 \oplus \vec{y}_1$, and sends $\vec{y}$ to P_0.
7. Each party P_σ outputs $\vec{y}$.

[a] Because C has depth k and each of its gate has fan-in at most 2, each output value only depends on at most 2^k inputs. Without loss of generality we can assume each output depends on exactly 2^k (by allowing for trivial "dependencies").

Fig. 8: **Secure Computation of Low-Depth Circuits from corrSPIR**

Theorem 16. *If* corrSPIR *is a two-round batch* SPIR *protocol with correlated "mix and match" queries, then Π_{2PC} from fig. 8 securely computes the randomized functionality $(\vec{x}_0, \vec{x}_1) \mapsto \{(\vec{r}, C(\vec{x}_0 \oplus \vec{x}_1) \oplus \vec{r}) \colon \vec{r} \xleftarrow{\$} \{0,1\}^m\}$ in the presence of a semi-honest adversary corrupting (at most) one of the two parties.*

Proof. First note that by lemma 13, there indeed exists a family of permutations $(\pi_j)_{j\in[k]}$ such as the one required by Π_{2PC} (and that such a family can be found in expected constant time by simply sampling random permutations and testing for the "T-balanced" property). Therefore the protocol is well-defined. Correctness follows from correctness of corrSPIR, with the observation that if $(\alpha_1, \ldots, \alpha_k) := \mathsf{MixAndMatch}_C(\vec{x}_1)$ then by construction $\mathtt{DB}_j[\alpha_j] = C_j(\vec{x}_1 \oplus \vec{x}_0) \oplus \vec{y}_0[j]$. It follows that $(\mathtt{DB}_1[\alpha_1], \ldots, \mathtt{DB}_m[\alpha_m]) \oplus \vec{y}_0 = C(\vec{x}_1 \oplus \vec{x}_0)$. Π_{2PC} essentially consists in a single call to Π_{corrSPIR}, and security follows from the security of corrSPIR via a standard hybrid argument. □

Our first main theorem follows from the combination of theorem 9 (which instantiates dec-OT from QR+LPN), lemma 11 (which provides a construction of rep-OT from dec-OT), theorem 15 (which provides a construction of corrSPIR from rep-OT), and theorem 16 (which provides a secure computation protocol from corrSPIR).

Main Theorem 1 (Sublinear Secure Computation from QR + LPN)**.** *Assume the* QR *assumption and the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim}, \mathsf{num}, \rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = (n+m)^{1/3} \cdot \mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{\varepsilon-1}$ *(for some constant $\varepsilon < 1$). Then for any n-input m-output boolean circuit C of size s and depth k, there is a two-party protocol for securely computing C using only* $\mathcal{O}(n + m + 2^{k+2^k} \cdot \mathsf{polylog}(n) \cdot \mathsf{poly}(\lambda) \cdot ((n+m)^{2/3} + (n+m)^{(1+2\varepsilon)/3}))$ *bits of communication, and computation* $\mathsf{poly}(\lambda, 2^{2^k})$.

The numbers are obtained by setting $\mathsf{dim} = \mathsf{poly}(\lambda)$, $\ell = (n+m)^{1/3}$, and $\mathsf{num} = \ell^2 \cdot \mathsf{dim}$ in the statement of Theorem 9. The $\mathsf{polylog}(n)$ term stems from the bounded query repetition property (Section 4.4) and the 2^{2^k+k} stems from the fact that the batch computational SPIR does, in essence, apply $\approx 2^{2^k}$ instances of the sender OT function in each interative compression step, and there are 2^k such steps. For example, absorbing the $\mathsf{polylog}(n)$ term in the $\mathsf{poly}(\lambda)$ term and setting $\varepsilon = 1/2$, and $k = (\log\log s)/4$ (implying $2^{k+2^k} \ll \sqrt{s}$), we get corollary 17:

Corollary 17 (Sublinear Secure Computation of $\log\log$-Depth Circuits). *Assume the* QR *assumption and the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim},\mathsf{num},\rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = (n+m)^{1/3}\cdot\mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{-1/2}$*. Then for any n-input m-output boolean circuit C of polynomial size s and depth* $\log\log s/4$*, there is a two-party protocol for securely computing C using only* $\mathcal{O}(n+m+\sqrt{s}\cdot\mathsf{poly}(\lambda)\cdot(n+m)^{2/3})$ *bits of communication, and polynomial computation.*

4.6 Extension to Layered Circuits

Layered circuits are boolean circuits whose gates can be arranged into layers such that any wire connects adjacent layers. It is well-known from previous works [BGI16, Cou19, CM21] that sublinear protocols for low-depth circuits translate to sublinear protocols for general layered circuits: the parties simply cut the layered circuit into low-depth "chunks", and securely evaluate it chunk-by-chunk. For each chunk, a sublinear secure protocol is invoked to compute the low-depth function which maps shares of the values on the first layer to shares of the values on the first layer of the next chunk. In particular, we get as a corollary from Theorem 1:

Corollary 18 (Sublinear Secure Computation of Layered Circuits). *Assume the* QR *or* DDH *assumption. Then for any n-input m-output layered boolean circuit C of polynomial size s and depth d, and any k, assuming the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim},\mathsf{num},\rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = ((s/d)^2/2^{2^k})^{1/3}\cdot\mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{-1/2}$*, there is a two-party protocol for securely computing C using communication*

$$\mathcal{O}\left(n+m+\frac{d}{k}\cdot\left(2^{2^k}\cdot\frac{s}{d}\right)^{2/3}\cdot\mathsf{poly}(\lambda)+\frac{s}{k}\right),$$

and computation $\mathsf{poly}(\lambda, 2^{2^k})$*.*

In the above corollary, "layered" refers to layered circuits whose inputs are on the first layer; this type of layered circuit is sometimes called *synchronous* in the literature. Furthermore, the corollary also uses a slightly optimized choice of parameters (compared to a naive application of Theorem 1): we set ℓ such that $\ell^2 = 2^{2^k+k}\cdot\mathsf{num}$ and $\mathsf{num} = ((s/d)^2/2^{2^k})^{1/3}\cdot\mathsf{poly}(\lambda)$ in the statement of Theorem 9. The above leads to a sublinear secure computation protocol for layered circuit whenever the circuit is not too "tall and skinny", *i.e.*, d is not too close to s. For example:

Corollary 19 ($s/\log\log s$-Secure Computation of Layered Circuits). *Assume the* QR *or* DDH *assumption. Then for any n-input m-output layered boolean circuit C of polynomial size s and depth d, for any constant $\varepsilon \in (0,1)$, assuming the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim},\mathsf{num},\rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = ((s/d)^2/s^{\varepsilon})^{1/3}\cdot\mathsf{poly}(\lambda)$*, and noise rate* $\rho = \mathsf{num}^{-1/2}$*, there is a two-party protocol for securely computing C using communication*

$$\mathcal{O}\left(n+m+d^{1/3}\cdot s^{2(1+\varepsilon)/3}\cdot\mathsf{poly}(\lambda)+\frac{s}{\log\log s}\right),$$

and computation $\mathsf{poly}(\lambda)$*.*

The above is sublinear in s as soon as $d = o(s^{1-\varepsilon}/\mathsf{poly}(\lambda))$.

5 Polylogarithmic PIR from CDH

A private information retrieval is a two-party protocol between a server S holding a string z (the database) and a client C holding an integer i, where only the client receives an output. The security parameter λ and the length $n(\lambda) = \mathsf{poly}(\lambda) = |z|$ of the server database are a common (public) input. We let $\mathsf{View}_S(\lambda, z, i)$ denote the view of S during its interaction with C on respective inputs (z, i) with common input $(\lambda, n = |z|)$, and by $\mathsf{Out}_C(\lambda, z, i)$ the output of C after the interaction.

Definition 20 (Private Information Retrieval). *A private information retrieval for database size $n = n(\lambda)$ (n-PIR) is an interactive protocol between a PPT server S holding a string $z \in \{0,1\}^n$ and a PPT client C holding an index $i \leq n$ which satisfies the following properties:*

– ***Correctness:*** *there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, $z \in \{0,1\}^n$, $i \in [n]$:*

$$\Pr[\mathsf{Out}_C(\lambda, z, i) = z_i] \geq 1 - \mu(\lambda).$$

– ***Security:*** *there exists a negligible function μ such that for every PPT adversary $\mathcal{A}$, large enough $\lambda \in \mathbb{N}$, $(i,j) \in [n]^2$, and $z \in \{0,1\}^n$:*

$$|\Pr[\mathcal{A}(1^{\lambda+n}, \mathsf{View}_S(\lambda, z, i)) = 1] - \Pr[\mathcal{A}(1^{\lambda+n}, \mathsf{View}_S(\lambda, z, j)) = 1]| \leq \mu(\lambda, n).$$

– ***Efficiency:*** *A PIR is* polylogarithmic *if its communication complexity $c(\lambda, n)$, measured as the worst-case number of bits exchanged between S and C (over their inputs (z, i) and their random coins), satisfies $c(\lambda, n) = \mathsf{poly}(\lambda, \log n)$.*

Remark 21. In the above definition, the adversary runtime is allowed to depend polynomially on n, while the parameter λ controls the strength of the cryptographic assumption upon which the protocol relies. Therefore, whenever $n = n(\lambda)$ is subexponentially large in λ, we have to assume subexponential hardness of the underlying assumption. Furthermore, when n can be as large as 2^λ, the underlying cryptographic assumptions cannot be satisfied anymore, hence we always assume in particular that $n < 2^\lambda$. This implies that $\log n$ is always bounded above by λ, hence technically $c(\lambda, n) = \mathsf{poly}(\lambda, \log n)$ is always just $\mathsf{poly}(\lambda)$ (which can be equal to $\mathsf{poly}(\log n)$ when λ is polylogarithmic in n, *i.e.*, in the subexponential security regime). This technicality is present in all previous works on sublinear PIR and related primitives (including e.g. laconic PSI [ABD⁺21], which we will use in our construction) and is usually ignored. In line with this, we stick to the terminology "polylogarithmic PIR" which is standard.

In this section, we prove the following result:

Main Theorem 2. *Assuming the hardness of the computational Diffie-Hellman assumption against $\mathsf{poly}(n)$-time adversaries, there exists a polylogarithmic n-PIR protocol, with polylogarithmic client computation, and $O(\log n)$ rounds.*

5.1 Laconic Private Set Intersection

Definition 22 (Laconic PSI [ABD⁺21]). *An ℓPSI scheme $\mathsf{LPSI} = (\mathsf{Setup}, \mathsf{R}_1, \mathsf{S}, \mathsf{R}_2)$ is defined as follows:*

– $\mathsf{Setup}(1^\lambda)$*: Take as input a security parameter 1^λ and outputs a common reference string* crs.
– $\mathsf{R}_1(\mathsf{crs}, S_R)$*: takes as input a* crs *and a receiver set S_R. Outputs a first PSI message* psi_1 *and a state* st.
– $\mathsf{S}(\mathsf{crs}, S_S, \mathsf{psi}_1)$*: takes as input a* crs*, a sender set S_S, and a first PSI message* psi_1*. Outputs a second PSI message* psi_2.
– $\mathsf{R}_2(\mathsf{crs}, \mathsf{st}, \mathsf{psi}_2)$*: takes as input a* crs*, a state* st*, and a second PSI message* psi_2*. Outputs a set $\mathcal{X}$.*

An ℓPSI protocol satisfies the following properties:

– *Correctness: for every sets (S_R, S_S), given* $\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{psi}_1, \mathsf{st}) \xleftarrow{\$} \mathsf{R}_1(\mathsf{crs}, S_R)$*,* $\mathsf{psi}_2 \xleftarrow{\$} \mathsf{S}(\mathsf{crs}, S_S, \mathsf{psi}_1)$*, and* $\mathcal{X} \xleftarrow{\$} \mathsf{R}_2(\mathsf{crs}, \mathsf{st}, \mathsf{psi}_2)$*, it holds that $\mathcal{X} = S_R \cap S_S$ with probability 1.*
– *Security: the two-round protocol defined by* $\mathsf{LPSI} = (\mathsf{Setup}, \mathsf{R}_1, \mathsf{S}, \mathsf{R}_2)$ *implements the PSI functionality given on Figure 9 in the semi-honest model.*
– *Efficiency: there exists a fixed polynomial* poly *such that both the length of* psi_1 *and the running time of* S *are bounded by* $\mathsf{poly}(\lambda, \log |S_R|)$.

Functionality $\mathcal{F}_{\mathsf{psi}}$

Parameters: The PSI functionality $\mathcal{F}_{\mathsf{psi}}$ is parameterised with a universe $\mathcal{U}$.
Setup Phase: The functionality waits until it receives S_R with $S_R \subseteq \mathcal{U}$ from R. Ignores subsequent messages from R.
Send Phase: The functionality waits until it receives S_S with $S_S \subseteq \mathcal{U}$ from S. Sends $S_R \cap S_S$ to R. Ignores subsequent messages from R.

Fig. 9: PSI functionality $\mathcal{F}_{\mathsf{psi}}$

Lemma 23 (ℓPSI from CDH [ABD+21]). *Assuming the security of the computational Diffie-Hellman assumption against* $\mathsf{poly}(n)$*-time adversaries, there exists an ℓPSI protocol for receiver sets of size n with statistical receiver security and computational (semi-honest) sender security.*

Remark 24. Lemma 23 is a simplified version of the result of [ABD+21]: the work of [ABD+21] actually achieves the stronger functionality of *reusable* PSI (where the setup phase is associated to a session id sid, the send phase is associated to an sid and a subsession id ssid, and the functionality can be reused indefinitely), and securely realizes this functionality in the UC model.

5.2 From Laconic PSI to Half-PIR

We define the notion of half-PIR, first introduced in [BIP18] (under the name $\mathsf{Rand}\frac{1}{2}\mathsf{PIR}$). Informally, a half-PIR behaves as a regular PIR with probability 1/2; otherwise, correctness and security might not hold. The receiver gets notified when the scheme successfully worked as intended.

Definition 25 (Half-PIR). *A half-PIR protocol is defined as an n-PIR (Definition 20) where the correctness and security properties are modified as follows:*

- ***Correctness:*** *there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, $z \in \{0,1\}^n$, $i \in [n]$:*

$$\Pr[\mathsf{Out}_C(\lambda, z, i) = (z_i, \mathsf{success})] \geq 1/2 - \mu(\lambda).$$

- ***Security:*** *there exists a negligible function μ such that for every PPT adversary $\mathcal{A}$, large enough $\lambda \in \mathbb{N}$, $(i,j) \in [n]^2$, and $z \in \{0,1\}^n$, it holds that $|p_i - p_j| \leq \mu(n, \lambda)$, where for an integer $k \in [n]$, p_k denotes the* conditional *probability* $\Pr[\mathcal{A}(1^{\lambda+n}, \mathsf{View}_S(\lambda, z, k)) = 1 \mid \mathsf{Out}_C(\lambda, z, k)_2 = \mathsf{success}]$.

Below, we recall the definition of pseudorandom functions (PRFs), first introduced in the seminal work of [GGM84]. For simplicity, we restrict our attention to PRFs with key length and output length equal to the security parameter λ.

Definition 26 (Pseudorandom function [GGM84,NR95]). *A* pseudorandom function *with input size m is syntactically defined by a function family $\mathcal{F} = \{F_K : \{0,1\}^{m(\lambda)} \mapsto \{0,1\}^\lambda\}_{\lambda \in \mathbb{N}, K \in \{0,1\}^\lambda}$, where the output $F_K(x)$ can be computed from (K, x) in polynomial time, and which satisfies the following security property: for every $\lambda \in \mathbb{N}$ and every oracle PPT attacker $\mathcal{A}$, it holds that*

$$\left|\Pr_K[\mathcal{A}(1^\lambda)^{F_K(\cdot)} = 1] - \Pr_R[\mathcal{A}(1^\lambda)^{R(\cdot)} = 1]\right| \leq \mathrm{negl}(\lambda),$$

where $K \xleftarrow{\$} \{0,1\}^\lambda$, and $R\colon \{0,1\}^m \mapsto \{0,1\}^\lambda$ is a truly random function. Furthermore, we say that the PRF is $T(\lambda)$-secure *if the above inequality still holds when $\mathcal{A}$ is additionally given 1^T as input.*

Protocol description. The high level intuition of the protocol is relatively simple: the client and the server agree on a PRF key K. Then, for each entry z_i of its database z, the server adds $F_K(i||z_i)$ to a set S_R. Now, the client with input i will randomly pick a bit b, and use a laconic PSI protocol (playing the role of the sender) to let the server (playing the receiver) learn whether $y = F_K(i||b)$ belongs to S_R. Eventually, the server tells the client whether the intersection size was 1 or 0. Observe that:

- If $b = z_i$, the server learns y, and can recover i. This corresponds to a failure of the protocol.
- If $b \neq z_i$, however, it holds with high probability that $y \notin S_R$. In this case, the server only learns that $S_R \cap \{y\} = \varnothing$, but does not learn the value of y. However, learning this information still tells the client that $b \neq z_i$, allowing him to recover z_i.

By the above, correctness and security fail either when $b = z_i$, which happens with probability exactly 1/2, or if a collision happens in ther database (*i.e.* $F_K(i||b)$ is equal to $F_K(j||z_j)$ for some $j \neq i$). Under the security of the PRF, the latter can be shown to happen at most with negligible probability. The full protocol is represented in fig. 10.

Half-PIR from Laconic PSI and PRF.

Parameters: The protocol is parameterised with a security parameter λ, and a database size $n = n(\lambda) \leq 2^\lambda \cdot \mathrm{negl}(\lambda)$. $\{F_K\}_{K \in \{0,1\}^\lambda}$ is a family of $n(\lambda)$-secure PRFs with input size $m = \log n + 1$. The protocol operates in the $\mathcal{F}_{\mathsf{psi}}$-hybrid model, where the universe $\mathcal{U}$ is defined as $\{0,1\}^\lambda$. The server holds an input string $z \in \{0,1\}^n$ and a the client holds an index $i \leq n$.

Protocol: The protocol operates in three steps.

1. The server picks a random PRF key $K \xleftarrow{\$} \{0,1\}^\lambda$ and sends it to the client. The client samples a uniformly random bit $b \xleftarrow{\$} \{0,1\}$, and sets $y \leftarrow F_K(i||b)$.
2. The server constructs the set $S_R = \{F_K(1||z_1), \cdots, F_K(n||z_n)\}$, and queries (sid, S_R) to $\mathcal{F}_{\mathsf{psi}}$, playing the role of the receiver. The client constructs the set $S_S = \{y\}$ and queries S_S to $\mathcal{F}_{\mathsf{psi}}$, playing the role of the sender. The server receives $S_R \cap S_S$.
3. The server indicates whether $S_R \cap S_S$ is empty by sending a bit to the client. If $S_R \cap S_S$ is empty, the client outputs $(1 - b, \mathsf{success})$; otherwise, the client outputs (b, fail).

Fig. 10: Half-PIR from Laconic PSI and PRF.

Security analysis. We prove that the above protocol satisfies computational correctness and perfect security in the $\mathcal{F}_{\mathsf{psi}}$-hybrid model, as defined in Definition 25. For computational correctness,

$$\begin{aligned}\Pr[\mathsf{Out}_C(\lambda, z, i) = (z_i, \mathsf{success})] &\leq \Pr[S_R \cap S_S \text{ is empty}] \\ &= \Pr[b \neq z_i \ \wedge \ F_K(i||b) \notin S_R] \\ &= (1/2) \cdot \Pr[F_K(i||1 - z_i) \notin \{F_K(j||z_j)\}_{j \in [n]}],\end{aligned}$$

where the last equality follows from the fact that b is an independent random bit, hence $\Pr[b \neq z_i] = 1/2$. Since $i||1 - z_i$ is not equal to any of the $j||z_j$ for $j = 1$ to n, the last probability is just the probability of a PRF evaluation not colliding with n other PRF evaluations. Now, for any fixed list of $n + 1$ distinct inputs $(x_0, \cdots, x_n)$, it holds by a straightforward union bound that $\Pr[R(x_0) \in \{R(x_1), \cdots, R(x_n)\}] \leq n/2^\lambda$, where the probability is taken over the choice of a uniformly random function $R : \{0,1\}^m \mapsto \{0,1\}^\lambda$. Since $n/2^\lambda = \mathrm{negl}(\lambda)$, it must therefore also hold that for any fixed list of $n + 1$ distinct inputs $(x_0, \cdots, x_n)$, $\Pr[F_K(x_0) \in \{F_K(x_1), \cdots, F_K(x_n)\}] \leq \mathrm{negl}(\lambda)$ as well. Indeed, assume for the sake of contradiction that this does not hold, and let us denote p this probability. We get the following contradiction to the assumption that $\{F_K\}_K$ is an $n(\lambda)$-secure PRF: the adversary $\mathcal{A}$ has the list $(x_0, \cdots, x_n)$ hardcoded in its description, queries the oracle on x_0 to x_n, obtaining answers $y_0 \cdots y_n$, and outputs 1 if $y_0 \in \{y_1, \cdots, y_n\}$. It is straightforward to see that the advantage of this adversary is $(p - n/2^\lambda)/2$, which is nonnegligible whenever p is nonnegligible. Therefore,

$$\Pr[\mathsf{Out}_C(\lambda, z, i) = (z_i, \mathsf{success})] = \frac{1}{2} \cdot (1 - \mathrm{negl}(\lambda)),$$

which concludes the proof of correctness. We now prove security: fix any two distinct client inputs (i, j). Then, for any $k \in \{i, j\}$,

$$\begin{aligned}&\Pr[\mathcal{A}(1^{\lambda+n}, \mathsf{View}_S(\lambda, z, k)) = 1 \mid \mathsf{Out}_C(\lambda, z, k)_2 = \mathsf{success}] \\ =&\Pr[\mathcal{A}(1^{\lambda+n}, K, S_R \cap S_S) = 1 \mid \mathsf{Out}_C(\lambda, z, k)_2 = \mathsf{success}] \\ =&\Pr[\mathcal{A}(1^{\lambda+n}, K, \varnothing) = 1 \mid \mathsf{Out}_C(\lambda, z, k)_2 = \mathsf{success}],\end{aligned}$$

since the client output $\mathsf{success}$ only when learning that $S_R \cap S_S = \varnothing$. Since $(K, \varnothing)$ is independent of k, we therefore have

$$\begin{aligned}p_i &= \Pr[\mathcal{A}(1^{\lambda+n}, K, \varnothing) = 1 \mid \mathsf{Out}_C(\lambda, z, i)_2 = \mathsf{success}] \\ &= \Pr[\mathcal{A}(1^{\lambda+n}, K, \varnothing) = 1 \mid \mathsf{Out}_C(\lambda, z, j)_2 = \mathsf{success}] = p_j,\end{aligned}$$

hence $|p_i - p_j| = 0$. This concludes the analysis.

Instantiating the functionalities. Pseudorandom functions can be constructed from one-way functions [GGM84]. Instantiating the functionality $\mathcal{F}_{\mathsf{psi}}$ with the CDH-based laconic PSI protocol of [ABD$^+$21] involves communication and client computation $\mathsf{poly}(\lambda, \log |S_R|) = \mathsf{poly}(\lambda, \log n)$ (since $|S_R| = n$). Summing up, we have:

Lemma 27. *Assuming the hardness of the computational Diffie-Hellman assumption against* $\mathsf{poly}(n)$*-time adversaries, there exists a (constant-round) polylogarithmic half-PIR protocol for databases of size n (where the client computation is also polylogarithmic).*

5.3 From Polylogarithmic Half-PIR to Polylogarithmic PIR

We now describe a simple generic transformation from Half-PIR to PIR.

Random-index PIR. First, we recall the definition of *random-index* PIR from [GHM$^+$21]:

Definition 28 (Random-Index PIR). *A random-index PIR for database of size n is a two-party protocol between a server and a client which implements the random-index PIR functionality given on Figure 11 in the semi-honest model.*

Functionality $\mathcal{F}_{\mathsf{rpir}}$

Parameters: The functionality is parameterised with a database size n.
Server Message: The functionality waits until it receives $z \in \{0,1\}^n$ from the server.
Output: If the client is honest, sample $i \xleftarrow{\$} [n]$ and output (i, z_i) to the client. Otherwise, output z to the client.

Fig. 11: Random-index PIR functionality $\mathcal{F}_{\mathsf{rpir}}$.

Interestingly, random-index PIR was recently shown to imply full-fledged PIR, with only a logarithmic (in n) blowup in communication and rounds, in [GHM$^+$21]:

Lemma 29. *If there exists a random-index PIR protocol for databases of size n with communication complexity $c(\lambda, n)$ and round complexity $r(\lambda, n)$, then there exists an n-PIR protocol with communication complexity $O(c(\lambda, n) \cdot \log n)$ and round complexity $O(r(\lambda, n) \cdot \log n)$.*

The elegant construction proceeds recursively: the client with input i and the server with input z first execute a random-index PIR, where the client receices (i^*, z_{i^*}). Then, the client sends $\alpha \leftarrow i \oplus i^*$ to the server. The server divides z into $n/2$ (unordered) pairs $\{z_j, z_{j\oplus\alpha}\}$, and computes for each pair the value $z'_{\{j,j\oplus\alpha\}} = z_j \oplus z_{j\oplus\alpha}$. Note that $|z'| = n/2$. Furthermore, the client knows $z_{i^*} = z_{i\oplus\alpha}$; therefore, given $z'_{\{i,i\oplus\alpha\}}$, the client can retrieve z_i as $z'_{\{i,i\oplus\alpha\}} \oplus z_{i^*}$. The server updates its input to z' and the client updates its input to be the index i' of the pair $\{i, i \oplus \alpha\}$. The above procedure reduces the task of performing an n-PIR to that of performing an $n/2$-PIR, using a single invocation of a random-index PIR, and an additional client-to-server message of length $\log n$. After a logarithmic number of such steps, the database size $|z|$ is reduced to $O(1)$, and can be sent directly to the client with constant communication. We refer to [GHM$^+$21] for a formal proof of Lemma 29.

From half-PIR to random-index PIR. By the above, constructing PIR from half-PIR is reduced to constructing random-index PIR from half-PIR. The latter, however, is straightforward: the client and the server can simply execute a half-PIR, where the client picks its input uniformly at random. At the end of the protocol, if the client receives fail, both parties simply restart the protocol. By the correctness of the half-PIR, a successful execution will happen after an expected $O(1)$ number of restarts. Below, we describe a slight variant of this where the client runs λ half-PIRs in parallel, and outputs the lexicographically first successful output.

Random-Index PIR from Half-PIR.

Parameters: The protocol is parameterised with a security parameter λ, and a database size $n = n(\lambda) \leq 2^\lambda \cdot \mathrm{negl}(\lambda)$. The server holds an input string $z \in \{0,1\}^n$; the client has no input.

Protocol: The client samples λ uniformly random integers $(i_1, \cdots, i_\lambda) \xleftarrow{\$} [n]^\lambda$. The client and the server run in parallel λ instances of a half-PIR protocol with respective client inputs i_j and server input z. The client receives outputs $\mathsf{Out}_C(\lambda, z, i_j)$.

Output: The client sets j^* to be the lexicographically first j such that $\mathsf{Out}_C(\lambda, z, i_j) = (z_j, \mathsf{success})$ for some bit z_j. The client outputs $(z_{j^*}, \mathsf{success})$. If there is no such j, the client outputs $\perp$ instead.

The security analysis of the protocol is straightforward: if the client is corrupted, the simulator Sim queries $\mathcal{F}_{\mathsf{rpir}}$ on its behalf, gets the full server input z, and emulates the server honestly. If the server is corrupted, Sim gets (i, z_i) from the functionality, picks $(i_1, \cdots, i_\lambda) \xleftarrow{\$} [n]^\lambda$, and emulates honestly the client in λ parallel runs of the half-PIR protocol. Then, it outputs (i, z_i).
Assume that one of the runs is successful, and let j^* be the lexicographically first such run. By the security of the half-PIR, the views of the server in this run with client input i or j^* are indistinguishable. Since i is uniformly random by the functionality of $\mathcal{F}_{\mathsf{rpir}}$, the simulation is therefore indistinguishable from an honest run of the protocol. If no run is successful, however, the simulation fails. However, since a run is successful with probability at least $1/2 - \mathrm{negl}(\lambda)$ by the correctness of half-PIR, the probability of this event is $(1/2 + \mathrm{negl}(\lambda))^\lambda$, which is negligible. This concludes the analysis.
Combining this protocol with Lemma 29, we get:

Lemma 30. *If there exists a half-PIR protocol for databases of size n with communication complexity $c(\lambda, n)$ and round complexity $r(\lambda, n)$, then there exists an n-PIR protocol with communication complexity $O(\lambda \cdot c(\lambda, n) \cdot \log n)$ and round complexity $O(r(\lambda, n) \cdot \log n)$.*

Remark 31. The work of [BIP18] also showed that sublinear half-PIR implies sublinear PIR. However, the end result was much weaker: their result showed that *polylogarithmic* half-PIR implies *slightly sublinear* (*i.e.*, $O(n/\mathsf{poly}(\log n))$) n-PIR, using much more involved tools, namely, query-efficient locally decodable codes. Our simple reduction, combined with the result of [GHM+21], strongly strengthens their result, which has independent applications to the study of the limits of practical sublinear secure computation.

Putting together Lemmata 27 and 30 finishes the proof of Theorem 2.

Acknowledgments

Elette Boyle and Pierre Meyer were supported by AFOSR Award FA9550-21-1-0046, a Google Research Award, and ERC Project HSS (852952). Geoffroy Couteau was supported by the ANR SCENE.

References

ABD+21. Navid Alamati, Pedro Branco, Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Sihang Pu. Laconic private set intersection and applications. In *Theory of Cryptography Conference*, pages 94–125. Springer, 2021.

ADOS22. Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. *Cryptology ePrint Archive*, 2022.

AIK09. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. *Journal of Cryptology*, 22(4):429–469, October 2009.

AJL+12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

BBDP22. Zvika Brakerski, Pedro Branco, Nico Döttling, and Sihang Pu. Batch-ot with optimal rate. *To appear at Eurocrypt 2022*, 2022.

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$^+$20. 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, Durham, NC, USA, November 16–19, 2020. IEEE Computer Society Press.

BFKL94. Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

BFKR91. Donald Beaver, Joan Feigenbaum, Joe Kilian, and Phillip Rogaway. Security with low communication overhead. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology – CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 62–76, Santa Barbara, CA, USA, August 11–15, 1991. Springer, Heidelberg, Germany.

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

BGI17. Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 163–193, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press.

BI05. Omer Barkol and Yuval Ishai. Secure computation of constant-depth circuits with applications to database search problems. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 395–411, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

BIP18. Elette Boyle, Yuval Ishai, and Antigoni Polychroniadou. Limits of practical sublinear secure computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 302–332, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

BLSV18. Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 535–564, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 11–19, Chicago, IL, USA, May 2–4, 1988. ACM Press.

CG97. Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In *29th Annual ACM Symposium on Theory of Computing*, pages 304–313, El Paso, TX, USA, May 4–6, 1997. ACM Press.

CGKS95. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science*, pages 41–50, Milwaukee, Wisconsin, October 23–25, 1995. IEEE Computer Society Press.

Cha04. Yan-Cheng Chang. Single database private information retrieval with logarithmic communication. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04: 9th Australasian Conference on Information Security and Privacy*, volume 3108 of *Lecture Notes in Computer Science*, pages 50–61, Sydney, NSW, Australia, July 13–15, 2004. Springer, Heidelberg, Germany.

CLTV15. Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th*

Theory of Cryptography Conference, Part II, volume 9015 of *Lecture Notes in Computer Science*, pages 468–497, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany.

CM21. Geoffroy Couteau and Pierre Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 842–870, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

CMS99. Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

Cou19. Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 473–503, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

DFH12. Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 54–74, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

DG17a. 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.

DG17b. Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 537–569, Santa Barbara, CA, USA, August 20–24, 2017. 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.

DNNR17. Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The TinyTable protocol for 2-party secure computation, or: Gate-scrambling revisited. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 167–187, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

FGJI17. Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith III. Homomorphic secret sharing from paillier encryption. In *Provable Security - 11th International Conference, ProvSec 2017, Xi'an, China, October 23-25, 2017, Proceedings*, volume 10592, pages 381–399, 2017.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

GGM84. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press.

GHM$^+$21. Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. Random-index pir and applications. In *Theory of Cryptography Conference*, pages 32–61. Springer, 2021.

GM82. Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th Annual ACM Symposium on Theory of Computing*, pages 365–377, San Francisco, CA, USA, May 5–7, 1982. ACM Press.

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

IKM$^+$13. Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 600–620, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany.

IP07. Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

JLS22. Ayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from lpn over $\mathbb{F}_p$, dlin, and prgs in nc^0. *To appear at Eurocrypt 2022*, 2022.

Kil00. Joe Kilian. More general completeness theorems for secure two-party computation. In *32nd Annual ACM Symposium on Theory of Computing*, pages 316–324, Portland, OR, USA, May 21–23, 2000. ACM Press.

KO97. Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science*, pages 364–373, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

Lip05. Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC 2005: 8th International Conference on Information Security*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328, Singapore, September 20–23, 2005. Springer, Heidelberg, Germany.

NN01. Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *33rd Annual ACM Symposium on Theory of Computing*, pages 590–599, Crete, Greece, July 6–8, 2001. ACM Press.

NR95. Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. In *36th Annual Symposium on Foundations of Computer Science*, pages 170–181, Milwaukee, Wisconsin, October 23–25, 1995. IEEE Computer Society Press.

OS07. Rafail Ostrovsky and William E. Skeith III. A survey of single-database private information retrieval: Techniques and applications (invited talk). In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 393–411, Beijing, China, April 16–20, 2007. Springer, Heidelberg, Germany.

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, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 678–708, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

RS21. Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 687–717, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

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 Rate-1 Batch OT Construction, Adapted from [BBDP22]

For completeness, we provide a full description of the 2-round rate-1 batch OT construction from [BBDP22, Section 7], which we cast as decomposable. Their construction is centered around packed linearly homomorphic encryption, whose definition we recall in definition 32. Our contribution is to observe that if this packed LHE satisfies an additional property of "shrunken ciphertext decomposability", which we define in definition 33 and show in lemma 35 to be a property held by many concrete instantiations, then this two-round batch oblivious transfer is in fact decomposable.

A.1 Decomposable Packed Linearly Homomorphic Encryption

We recall in definition 32 the definition of packed linearly homomorphic encryption, and introduce in definition 33 the notion of *decomposability* for such an encryption scheme.

Definition 32 ((Packed) Linearly Homomorphic Encryption, [BBDP22]). *A* packed linearly homomorphic encryption *(LHE) scheme $\mathcal{LHE}$ over a finite group $\mathbb{G}$ is a tuple of* p.p.t. *algorithms $\mathcal{LHE} = (\mathcal{LHE}.\mathsf{KeyGen}, \mathcal{LHE}.\mathsf{Enc}, \mathcal{LHE}.\mathsf{Shrink}, \mathcal{LHE}.\mathsf{DecShrink})$ with the following syntax and properties:*

- $\mathsf{KeyGen}(1^\lambda, k)$*: On input a security parameter 1^λ and a plaintext length $k \in \mathbb{N}^\star$,* KeyGen *outputs a public key* pk *and a secret key* sk*. The size of the public key output by* $\mathsf{KeyGen}(1^\lambda, k)$ *is bounded by* $k \cdot \mathsf{poly}(\lambda)$*.*
- $\mathsf{Enc}(\mathsf{pk}, \vec{m} = (m_1, \ldots, m_k))$*: On input a public key* pk *and a message* $\vec{m} = (m_1, \ldots, m_k) \in \mathbb{G}^k$*,* Enc *outputs a ciphertext* ct*.*
- $\mathsf{Eval}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))$*: On input a public key* pk*, a linear function $f \colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, and a batch of ℓ ciphertexts* $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$*,* Eval *outputs a ciphertext* $\tilde{\mathsf{ct}}$*.*
- $\mathsf{Shrink}(\mathsf{pk}, \mathsf{ct})$*: On input a public key* pk *and a ciphertext* ct*,* Shrink *outputs a shrunken ciphertext* ct'*.*
- $\mathsf{DecShrink}(\mathsf{sk}, \mathsf{ct})$*: On input a secret key* sk *and a shrunken ciphertext* ct*,* $\mathsf{DecShrink}$ *outputs a message* $\vec{m}$*.*
- Correctness. *For any $\ell \in \mathbb{N}$, any messages $\vec{m}_1, \ldots, \vec{m}_\ell \in \mathbb{G}^k$, and any linear function $f \colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$,*

$$\Pr\left[f(\vec{m}_1, \ldots, \vec{m}_\ell) = \tilde{m} \ : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) \\ \tilde{m} \xleftarrow{\$} \mathsf{DecShrink}(\mathsf{sk}, \tilde{\mathsf{ct}}) \end{array} \right] = 1$$

where $\mathsf{Eval\&Shrink}$ *is an additional algorithm defined for convenience: On input a public key* pk*, a linear function f, and a batch of ℓ ciphertexts* $(\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$*,* $\mathsf{Eval\&Shrink}$ *runs* $\tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Shrink}(\mathsf{pk}, \mathsf{Eval}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)))$ *and outputs the ciphertext* $\tilde{\mathsf{ct}}$*.*
- Semantic Security. *For all $\lambda \in \mathbb{N}$, $k = \mathsf{poly}(\lambda)$, and* p.p.t. *adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$,*

$$\Pr\left[b' = b \ : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ (\vec{m}_0, \vec{m}_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_0(\mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\} \\ \mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_b) \\ b' \xleftarrow{\$} \mathcal{A}_1(\mathsf{st}, \mathsf{ct}) \end{array} \right] \leq \mathrm{negl}(\lambda) \ .$$

- Compactness.
  - *For sufficiently large $k \in \mathbb{N}$, any $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Supp}(\mathsf{KeyGen}(1^\lambda, k))$, the size of the public key,* i.e. $|\mathsf{pk}|$*, is upper bounded by* $k \cdot \mathsf{poly}(n)$*.*
  - Rate-1. *For sufficiently large $k \in \mathbb{N}$, any linear function $f \colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, and any $(\vec{m}_1, \ldots, \vec{m}_\ell) \in (\mathbb{G}^k)^\ell$, for all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Supp}(\mathsf{KeyGen}(1^\lambda, k))$ and $\mathsf{ct}_i \in \mathsf{Supp}(\mathsf{Enc}(\mathsf{pk}, \vec{m}_i))$, $i \in [\ell]$:*

$$|\mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))| = |f(\vec{m}_1, \ldots, \vec{m}_\ell)| \cdot (1 + o(1)) = (k \cdot \log |\mathbb{G}|) \cdot (1 + o(1)) \ .$$

  When convenient, we will parse a rate-1 shrunken ciphertext as $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$, where $|\mathsf{ct}_0| = o(k)$ and $|\mathsf{ct}_1| = k$.

– Function Privacy. *There exists a simulator* $\mathsf{Sim}^{\text{fn-priv}}_{\mathcal{LHE}}$ *such that for all messages* $(\vec{m}_1, \ldots, \vec{m}_\ell) \in (\mathbb{G}^k)^\ell$ *and all linear functions* $f \colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, *for all adversaries* $\mathcal{A}$,

$$\left| \Pr\left[b = 1 \,:\, \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_i)_{i\in[\ell]}) \\ b \xleftarrow{\$} \mathcal{A}(\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}) \end{array} \right] - \Pr\left[b = 1 \,:\, \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Sim}^{\text{fn-priv}}_{\mathcal{LHE}}(\mathsf{pk}, f((\vec{m}_i)_{i\in[\ell]})) \\ b \xleftarrow{\$} \mathcal{A}(\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}) \end{array} \right] \right| \le \operatorname{negl}(\lambda) \;.$$

In other words, since $\mathsf{Sim}^{\text{fn-priv}}_{\mathcal{LHE}}$ *does not use the function* f *to compute* $\tilde{\mathsf{ct}}$, *no non-trivial information about it is leaked from* $\tilde{\mathsf{ct}}$.

Informally, a packed LHE scheme is *decomposable* if, given the secret key and a shrunken ciphertext (which has size $k + o(k)$) missing some or all of the last k bits (note that the set of erased positions is assumed to be known), there is a way to recover the corresponding subset of the (homomorphically evaluated) plaintext vector but no information about the rest of the plaintext. Note that if any bit of the shrunken ciphertext is dropped other than the last k, then there is no correctness guarantee on recovering any information about the plaintext. We formalise this notion in definition 33.

Definition 33 (Decomposable Linearly Homomorphic Encryption, LHE). *A packed linearly homomorphic encryption (LHE) scheme (definition 32)* $\mathcal{LHE} = (\mathcal{LHE}.\mathsf{KeyGen}, \mathcal{LHE}.\mathsf{Enc}, \mathcal{LHE}.\mathsf{Shrink}, \mathcal{LHE}.\mathsf{DecShrink})$ *over a finite group* $\mathbb{G}$ *is said to be* decomposable *if there exists a probabilistic polynomial time* partial decryption algorithm $\mathcal{LHE}.\mathsf{pDecShrink}$ *with the following syntax and properties:*

– Decomposability – Syntax. *On input a batch subset* $S \subseteq [k]$, *a secret key* sk, *a partial shrunken ciphertext* $\tilde{c} = (\tilde{c}_0, \tilde{c}_1)$ *where* $|\tilde{c}_0| = o(k \cdot \log |\mathbb{G}|)$ *and* $|\tilde{c}_1| = |S| \cdot \log |\mathbb{G}|$, $\mathcal{LHE}.\mathsf{pDecShrink}$ *outputs a partial message* $\tilde{m} \in \mathbb{G}^{|S|}$.

– Decomposability – Correctness. *For any* $\ell \in \mathbb{N}$, *any batch size* $k \in \mathbb{N}^\star$, *any messages* $\vec{m}_1, \ldots, \vec{m}_\ell \in \mathbb{G}^k$, *any linear function* $f \colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, *and any batch subset* $S \subseteq [k]$,

$$\Pr\left[(f(\vec{m}_1, \ldots, \vec{m}_\ell))[S] = \tilde{m} \,:\, \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) \\ \tilde{m} \xleftarrow{\$} \mathsf{pDecShrink}(S, \mathsf{sk}, (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S])) \end{array} \right] = 1 \;,$$

$$\text{where } I_S \coloneqq \bigcup_{i \in S} [(i-1)|\mathbb{G}|, i|\mathbb{G}|] \;.$$

– Decomposability – Security. *There exists an expected polynomial time simulator* $\mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}$ *such that for every* $\lambda \in \mathbb{N}^\star$, *any* $\ell \in \mathbb{N}$, *any batch size* $k \in \mathbb{N}^\star$, *any messages* $\vec{m}_1, \ldots, \vec{m}_\ell \in \mathbb{G}^k$, *any linear function* $f \colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, *and any batch subset* $S \subseteq [k]$,

$$\left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S]) \,:\, \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) \end{array} \right\} \overset{c}{\approx}$$

$$\left\{ (\mathsf{pk}, \mathsf{sk}, \mathsf{sim}_0, \mathsf{sim}_1) \,:\, \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ (\mathsf{sim}_0, \mathsf{sim}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, S, f, (f(\vec{m}_1, \ldots, \vec{m}_\ell))[S]) \end{array} \right\} \;,$$

$$\text{where } I_S \coloneqq \bigcup_{i \in S} [(i-1)|\mathbb{G}|, i|\mathbb{G}|] \;.$$

At a high level, function-privacy guarantees that, even given the secret key, a post-homomorphism shrunken ciphertext does not leak the function while decomposability guarantees that dropping selected parts of this ciphertext conceals information about the corresponding (post-homomorphism) plaintext. It makes intuitive sense that these properties should be achievable simultaneously, however this may not be clear *a priori* from the formalism since the simulator $\mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}$ in definition 33 is given as input the function. We nevertheless establish this fact in lemma 34.

Lemma 34 (Function-Private Decomposability). *If $\mathcal{LHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Shrink}, \mathsf{DecShrink})$ is a decomposable packed linearly homomorphic encryption scheme over $\mathbb{G}$, then there exists an expected polynomial time simulator $\mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}$ such that for every $\lambda \in \mathbb{N}^\star$, any $\ell \in \mathbb{N}$, any batch size $k \in \mathbb{N}^\star$, any messages $\vec{m}_1, \dots, \vec{m}_\ell \in \mathbb{G}^k$, any linear function $f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$, and any batch subset $S \subseteq [k]$,*

$$\left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S]) \ :\ \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \textit{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \dots, \mathsf{ct}_\ell)) \end{array} \right\} \overset{c}{\approx}$$

$$\left\{ (\mathsf{pk}, \mathsf{sk}, \mathsf{sim}_0, \mathsf{sim}_1) \ :\ \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ (\mathsf{sim}_0, \mathsf{sim}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, S, (f(\vec{m}_1, \dots, \vec{m}_\ell))[S]) \end{array} \right\} ,$$

$$\textit{where } I_S := \bigcup_{i \in S} [(i-1)|\mathbb{G}|, i|\mathbb{G}|] \ . \quad (2)$$

In particular, note that f is not given as input to the simulator.

Proof. Let $\mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}$ be defined as in definition 33 and consider the following algorithm $\mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}$: On input $(1^\lambda, \mathsf{pk}, k, S, y)$, parse pk so as to retrieve ℓ, define $y' \in \mathbb{G}^k$ as $y'[x] := (y[x]$ if $x \in S$, and 0 otherwise$)$, run $(\mathsf{ct}_0, \mathsf{ct}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, S, \mathsf{cst}_{y'}, y)$ where $\mathsf{cst}_{y'}$ is the constant function on $(\mathbb{G}^k)^\ell$ equal to y', and output $(\mathsf{ct}_0, \mathsf{ct}_1[I_S])$ where $I_S := \bigcup_{i \in S}[(i-1)|\mathbb{G}|, i|\mathbb{G}|]$. Let us now show that it satisfies the required property for lemma 34.

Let $\mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}$ be defined as in definition 32. Before we proceed, note that:

$$\forall f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k \text{ linear}, \forall \vec{m}_1, \dots, \vec{m}_\ell \in \mathbb{G}^k, \forall S \subseteq [k],$$

$$\left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S]) \ :\ \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \dots, \mathsf{ct}_\ell)) \end{array} \right\}$$

$$\overset{c}{\approx} \left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S]) \ :\ \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}(\mathsf{pk}, f(\vec{m}_1, \dots, \vec{m}_\ell)) \end{array} \right\} ,$$

$$\text{where } I_S := \bigcup_{i \in S} [(i-1)|\mathbb{G}|, i|\mathbb{G}|] \ . \quad (3)$$

Indeed, should there exist $f, \vec{m}_1, \dots, \vec{m}_\ell, S$ such that there existed a *p.p.t.* adversary $\mathcal{A}$ with non-negligeable advantage in distinguishing the two above distributions, then $\mathcal{A}'$ defined as $\mathcal{A}'(x_1, x_2, x_3, x_4) \xleftarrow{\$} \mathcal{A}(x_1, x_2, x_3, x_4[I_S])$ (where $I_S := \bigcup_{i \in S}[(i-1)|\mathbb{G}|, i|\mathbb{G}|]$) would distinguish the following two distributions with non-negligeable probability, thereby contradicting function-privacy:

$$\left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \ :\ \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \dots, \mathsf{ct}_\ell)) \end{array} \right\}$$

$$\text{and } \left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \ :\ \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}(\mathsf{pk}, f(\vec{m}_1, \dots, \vec{m}_\ell)) \end{array} \right\} .$$

We are now ready to prove that our candidate $\mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}$ indeed satisfies the requirements of eq. (2). Let $\lambda \in \mathbb{N}^\star$, $\ell \in \mathbb{N}$, $k \in \mathbb{N}^\star$, $\vec{m}_1, \dots, \vec{m}_\ell \in \mathbb{G}^k$, and $S \subseteq [k]$. Let $f\colon (\mathbb{G}^k)^\ell \to \mathbb{G}^k$ be a linear function. Define $y := (f(\vec{m}_1, \dots, \vec{m}_\ell))[S]$, and $y' \in \mathbb{G}^k$ as $y'[x] := (y[x]$ if $x \in S$, and 0 otherwise$)$. Let $\mathsf{cst}_{y'}$ denote the constant function on $(\mathbb{G}^k)^\ell$ equal to y'.

Observe that, if we define $I_S \coloneqq \bigcup_{i \in S}[(i-1)|\mathbb{G}|, i|\mathbb{G}|]$:

$$\left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S]) \ : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \vec{m}_i) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, f, (\mathsf{ct}_1, \dots, \mathsf{ct}_\ell)) \end{array} \right\}$$

$$\overset{c}{\approx} \left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S]) \ : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{fn\text{-}priv}}_{\mathcal{LHE}}(\mathsf{pk}, f(\vec{m}_1, \dots, \vec{m}_\ell)) \end{array} \right\}$$

$$\overset{c}{\approx} \left\{ (\mathsf{pk}, \mathsf{sk}, \tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1[I_S]) \ : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, f(\vec{m}_i)) \text{ for } i \in [\ell] \\ \tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}_0, \tilde{\mathsf{ct}}_1) \xleftarrow{\$} \mathsf{Eval\&Shrink}(\mathsf{pk}, g, (\mathsf{ct}_1, \dots, \mathsf{ct}_\ell)) \end{array} \right\}$$

$$\overset{c}{\approx} \left\{ (\mathsf{pk}, \mathsf{sk}, \mathsf{sim}_0, \mathsf{sim}_1) \ : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ (\mathsf{sim}_0, \mathsf{sim}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, S, g, (f(\vec{m}_1, \dots, \vec{m}_\ell))[S]) \end{array} \right\}$$

$$\equiv \left\{ (\mathsf{pk}, \mathsf{sk}, \mathsf{sim}_0, \mathsf{sim}_1) \ : \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, k) \\ (\mathsf{sim}_0, \mathsf{sim}_1) \xleftarrow{\$} \mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, S, (f(\vec{m}_1, \dots, \vec{m}_\ell))[S]) \end{array} \right\} .$$

The first two steps follow from eq. (3), the third from the definition of $\mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}$, and the fourth by how we defined $\mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}$. ☐

We recall in fig. 12 the construction of packed LHE under QR from [DGI+19, BBDP22] and note that it is decomposable.

dec-$\mathcal{LHE}$, Adapted from [DGI+19, BBDP22]

KeyGen: On input the security parameter 1^λ and a batch size k:

1. Choose two safe primes $p = 2p' + 1$ and $q = 2q' + 1$ where p', q' are primes and compute $N = pq$. Choose a generator g of $\mathbb{QR}_N$.
2. Sample $s \xleftarrow{\$} \mathbb{Z}^k_{\phi(N)/2}$ and compute $\vec{h} \leftarrow g^s$.
3. Output $\mathsf{pk} \leftarrow (N, g, \vec{h})$ and $\mathsf{sk} \leftarrow s$.

Enc: On input the public key pk and a batch of k messages $\vec{m} = (m_1, \dots, m_k)$:

1. Parse pk as $\mathsf{pk} = (N, g, \vec{h} = (h_1, \dots, h_k))$
2. Sample $r \xleftarrow{\$} \mathbb{Z}_{(N-1)/2}$. Compute $c_1 \leftarrow g^r$ and $c_{2,i} \leftarrow (-1)^{m_i} h_i^r$ for $i \in [k]$.
3. Output $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2 = (c_{2,1}, \dots, c_{2,k}))$.

Eval: On input a public key pk, an ℓ-input linear function f, and ℓ ciphertexts $(\mathsf{ct}_1, \dots, \mathsf{ct}_\ell)$:

1. Parse pk as $\mathsf{pk} = (N, g, \vec{h} = (h_1, \dots, h_k))$, f as $f(X_1, \dots, X_\ell) = \sum_{j=1}^{\ell} a_j \cdot X_j + \vec{b}$ where $a_1, \dots, a_\ell \in \{0, 1\}$ and $\vec{b} \in \{0, 1\}^k$; For $j \in [\ell]$, parse ct_j as $(c_{1,j}, \vec{c}_{2,j} = (c_{2,1,j}, \dots, c_{2,k,j}))$.
2. Sample $t \xleftarrow{\$} \mathbb{Z}_{(N-1)/2}$ and compute $\tilde{c}_1 \leftarrow g^t \cdot \prod_{j=1}^{\ell} c_{1,j}^{a_j}$ and $\tilde{c}_{2,i} \leftarrow h_i^t \cdot (-1)^{b_i} \cdot \prod_{j=1}^{\ell} c_{2,i,j}^{a_j}$, then set $\tilde{\mathsf{ct}} = (\tilde{c}_1, \tilde{c}_2 = (\tilde{c}_{2,1}, \dots, \tilde{c}_{2,k}))$.
3. Output $\tilde{\mathsf{ct}}$.

Shrink: On input a public key pk and a packed ciphertext ct:

1. Parse pk as $\mathsf{pk} = (N, g, \vec{h} = (h_1, \dots, h_k))$ and ct as $\mathsf{ct} = (c_1, c_2 = (c_{2,1}, \dots, c_{2,k}))$.
2. For $i \in [k]$, set $e_i \leftarrow 0$ if $c_{2,i} <_{\mathbb{J}_N} -c_{2,i}$ and $e_i \leftarrow 1$ otherwise.
3. Output $(c_1, (e_1, \dots, e_k))$.

DecShrink: On input a secret key sk and a shrunken ciphertext ct:

1. Parse ct as $\mathsf{ct} = (c_1, (e_1, \dots, e_k))$ where $\forall i \in [k], e_i \in \mathbb{G}$.
2. Parse sk as $\mathsf{sk} = (\mathsf{sk}_1, \dots, \mathsf{sk}_k)$.
3. For $i \in [k]$, set $\tilde{m}_i \leftarrow \mathsf{LEq}(c_1^{\mathsf{sk}_i}, (-1) \cdot c_1^{\mathsf{sk}_i})$.
4. Output $(\tilde{m}_1, \dots, \tilde{m}_k)$.

pDecShrink: On input a batch subset $S \subseteq [k]$, a secret key sk, a partial shrunken ciphertext $\tilde{c}$:

```
1. Parse $\tilde{c}$ as $\tilde{c} = (\tilde{c}_0, (e_i)_{i \in S})$ where $\forall i \in S, e_i \in \mathbb{G}$.
2. Parse $\mathsf{sk}$ as $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_k)$.
3. For $i \in S$, set $\tilde{m}_i \leftarrow \mathsf{LEq}(c_1^{\mathsf{sk}_i}, (-1) \cdot c_1^{\mathsf{sk}_i})$.
4. Output $(\tilde{m}_1, \ldots, \tilde{m}_k)$.
```

Fig. 12: Decomposable Packed Linearly Homomorphic Encryption from QR.

Lemma 35. *Assuming the Quadratic Residuosity assumption, the construction of fig. 12 is a decomposable packed linearly homomorphic encryption scheme.*

Proof. The above scheme was shown to be a circuit-private LHE by [DGI⁺19, BBDP22], so it only remains to show it is decomposable. Decomposable security follows from the fact that (with the notations of fig. 12) a partial ciphertext for batch subset S is of the form $\mathsf{ct} = (c_1, (e_i)_{i \in S})$, which can be observed to information-theoretically contain no information about $(m_i)_{i \in [N] \setminus S}$. Decomposable correctness follows from inspection of the "locality" of $\mathsf{DecShrink}$. $\square$

A.2 Two-Round co-PIR

We now recall the notion of *co-PIR* from [BBDP22, Section 6.1] (or *punctured OT* [BGI17]), which allows a receiver holding as input a set of indices S to interact with an input-free server in such a way that the sender obtains a pseudorandom string $\vec{y} \in \mathbb{Z}_q^m$ while the receiver gets $\vec{y}[[N] \setminus S]$ (all entries of $\vec{y}$ which are *not* in S).

Definition 36 (Two-Round co-PIR, [BBDP22]). *A two-round* co-PIR *scheme over $\mathbb{Z}_q$ (with poly-logarithmic communication complexity) is parameterised by an integer m where $m = \mathsf{poly}(\lambda)$, and is composed by a tuple of algorithms $\mathsf{copir} = (\mathsf{copir.Query}, \mathsf{copir.Send}, \mathsf{copir.Receive})$ with the following syntax and properties:*

- $\mathsf{Query}(1^\lambda, S)$: *On input the security parameter 1^λ and a set of indices $S \subseteq [m]$,* Query *outputs a receiver message* copir_R *and a private state* st.
- $\mathsf{Send}(\mathsf{copir}_R)$: *On input a receiver message* copir_R, Send *outputs a sender message* copir_S *and a string $\boldsymbol{y} \in \mathbb{Z}_q^m$.*
- $\mathsf{Dec}(\mathsf{copir}_S, \mathsf{st})$: *On input a sender message* copir_S *and a state* st, Dec *outputs a string $\tilde{\boldsymbol{y}} \in \mathbb{Z}_q^m$.*
- Correctness. *A co-pir scheme* copir *is said to be* correct *if for any $m = \mathsf{poly}(\lambda)$ and $S \subseteq [m]$,*

$$\Pr\left[\boldsymbol{y}[\overline{S}] = \tilde{\boldsymbol{y}}[\overline{S}] \colon \begin{array}{r} (\mathsf{copir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{Query}(1^\lambda, S) \\ (\mathsf{copir}_S, \boldsymbol{y}) \xleftarrow{\$} \mathsf{Send}(\mathsf{copir}_R) \\ \tilde{\boldsymbol{y}} \xleftarrow{\$} \mathsf{Receive}(\mathsf{copir}_S, \mathsf{st}) \end{array}\right] = 1 \ ,$$

where $\overline{S} = [m] \setminus S$.

- Receiver Security. *For all $m = \mathsf{poly}(\lambda)$, any subsets $S_1, S_2 \subseteq [m]$, any* p.p.t. *adversary $\mathcal{A}$,*

$$\left| \Pr\left[\mathcal{A}(k, \mathsf{copir}_R) = 1 \colon (\mathsf{copir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{Query}(1^\lambda, S_1)\right] - \Pr\left[\mathcal{A}(k, \mathsf{copir}_R) = 1 \colon (\mathsf{copir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{Query}(1^\lambda, S_2)\right] \right| \leq \operatorname{negl}(\lambda) \ .$$

- Sender Security. *For all $m = \mathsf{poly}(\lambda)$, any subset $S \subseteq [m]$, any* p.p.t. *adversary $\mathcal{A}$,*

$$\left| \Pr\left[\mathcal{A}(k, \mathsf{st}, \mathsf{copir}_S, \boldsymbol{y}_S) = 1 \colon \begin{array}{r} (\mathsf{copir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{Query}(1^\lambda, S) \\ (\mathsf{copir}_S, \boldsymbol{y}) \xleftarrow{\$} \mathsf{Send}(\mathsf{copir}_R, \boldsymbol{x}) \end{array}\right] - \Pr\left[\mathcal{A}(k, \mathsf{st}, \mathsf{copir}_S, \boldsymbol{y}'_S) = 1 \colon \begin{array}{r} (\mathsf{copir}_R, \mathsf{st}) \xleftarrow{\$} \mathsf{Query}(1^\lambda, S) \\ (\mathsf{copir}_S, \boldsymbol{y}) \xleftarrow{\$} \mathsf{Send}(\mathsf{copir}_R, \boldsymbol{x}) \\ \boldsymbol{y}'_S \xleftarrow{\$} \mathbb{Z}_q^{|S|} \end{array}\right] \right| \leq \operatorname{negl}(\lambda) \ .$$

– Compactness ("Polylogarithmic Communication"). *For all* $m = \mathsf{poly}(\lambda)$*, any subset* $S \subseteq [m]$*, any* $(\mathsf{copir}_R, \mathsf{st}) \in \mathsf{Supp}(\mathsf{Query}(1^\lambda, S))$*, and any* $(\mathsf{copir}_S, \boldsymbol{y}) \in \mathsf{Supp}(\mathsf{Send}(\mathsf{copir}_R))$*, it holds that* $|\mathsf{copir}_R|, |\mathsf{copir}_S| = |S| \cdot \mathsf{polylog}(m) \cdot \mathsf{poly}(\lambda)$.

Lemma 37 (Instantiation of co-PIR [BBDP22]). *Assuming the Quadratic Residuosity assumption, there exists two-round polylogarithmic co-PIR over* $\{0,1\}$.

A.3 Decomposable OT from Decomposable LHE

For the sake of more unified notations with the construction of rate-1 OT from [BBDP22, Section 7], we depart in this section from definition 20, and take two-round single-server private information retrieval with (polylogarithmic communication) to be a tuple of algorithms $\mathsf{PIR} = (\mathsf{PIR.Query}, \mathsf{PIR.Send}, \mathsf{PIR.Receive})$.

dec-OT

Parameters: Batch number $k = \ell \cdot t$; Exact LPN error τ; A constant $\epsilon \in (0,1)$ tied to the hardness of one of the underlying LPN assumptions.
Requires:

– $\mathcal{LHE} = (\mathcal{LHE}.\mathsf{KeyGen}, \mathcal{LHE}.\mathsf{Enc}, \mathcal{LHE}.\mathsf{Eval}, \mathcal{LHE}.\mathsf{Shrink}, \mathcal{LHE}.\mathsf{DecShrink})$ is a decomposable packed linearly homomorphic encryption scheme (with partial decryption algorithm $\mathsf{pDecShrink}$) with plaintext space $\{0,1\}^\ell$ and equipped with a post-homomorphism shrinking procedure $\mathcal{LHE}.\mathsf{Shrink}$ which converts ciphertexts into a rate 1 representation.
– $\mathsf{copir} = (\mathsf{copir.Query}, \mathsf{copir.Send}, \mathsf{copir.Receive})$ is a two-round polylogarithmic co-PIR scheme over $\{0,1\}$ and parameterised by a database size of t.
– $\mathsf{PIR} = (\mathsf{PIR.Query}, \mathsf{PIR.Send}, \mathsf{PIR.Receive})$ is a two-round polylogarithmic PIR scheme over $\{0,1\}$.

$\mathsf{dec\text{-}OTR}$: On input the security parameter 1^λ and a vector of selection bits $\vec{b} = (b_1, \ldots, b_k) \in \{0,1\}^k$:

```
1. Parse $\vec{b}$ as $\vec{b} = (\vec{b}_1, \ldots, \vec{b}_\ell)$ where each $\vec{b}_i \in \{0,1\}^t$ is a block of size $t$ .
2. Choose $\boldsymbol{A} \xleftarrow{\$} \{0,1\}^{n \times t}$ uniformly at random, and sample $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathcal{LHE}.\mathsf{KeyGen}(1^\lambda, \ell)$ .
3. For $i = 1, \ldots, \ell$:
   (a) Sample $\vec{s}_i \xleftarrow{\$} \{0,1\}^n$, and $\vec{e}_i \xleftarrow{\$} \mathsf{HW}_\tau(\{0,1\}^t)$ (Uniformly random $\tau$-sparse length-$t$ vector)
   (b) Compute $\vec{c}_i \leftarrow \vec{s}_i \cdot \boldsymbol{A} + \vec{e}_i + \vec{b}_i$
   (c) Set $\boldsymbol{S}_i \leftarrow \mathsf{SingleRowMatrix}(\ell, n, i, \vec{s}_i)$
   (d) Compute a matrix-ciphertext $\mathbf{ct}_i \xleftarrow{\$} \mathcal{LHE}.\mathsf{Enc}(\mathsf{pk}, \boldsymbol{S}_i)$
   (e) Set $J_i = \mathsf{supp}(\vec{e}_i)$
   (f) Compute $(\mathsf{copir}_{R,i}, \mathsf{st}_i) \leftarrow \mathsf{coPIR.Query}(J_i)$
4. Set $\mathsf{otr} \leftarrow (\mathsf{pk}, \boldsymbol{A}, \{\mathsf{ct}_i, \vec{c}_i, \mathsf{copir}_{R,i}\}_{i \in [\ell]}, \{q_{i,j}\}_{i \in [\ell], j \in [t]})$
5. Set $\mathsf{st} \leftarrow (\mathsf{sk}, \{\mathsf{st}_i, J_i\}_{i \in [\ell]}, \{\hat{\mathsf{st}}_{i,j}\}_{i \in [\ell], j \in [t]})$
6. Output $(\mathsf{otr}, \mathsf{st})$
```

$\mathsf{dec\text{-}OTS}$: On input the security parameter 1^λ, a receiver message otr, a database $\vec{m} \in \{0,1\}^{2k}$:

```
1. Parse $\mathsf{otr}$ as $\mathsf{otr} = (\mathsf{pk}, \boldsymbol{A}, \{\mathsf{ct}_i, \vec{c}_i, \mathsf{copir}_{R,i}\}_{i \in [\ell]}, \{q_{i,j}\}_{i \in [\ell], j \in [t]})$
2. Parse $\vec{m}$ as $\vec{m} = ((m_{0,i,j}, m_{1,i,j}))_{i \in [\ell], j \in [t]}$ and set $\vec{m}_{b,i} \leftarrow (m_{b,i,1}, \ldots, m_{b,i,t}) \in \{0,1\}^t$.
3. For $i = 1, \ldots, \ell$:
   (a) $(\vec{y}_i, \mathsf{copir}_S) \xleftarrow{\$} \mathsf{coPIR.Send}(\mathsf{copir}_{R,i})$ where $\vec{y}_i \leftarrow (y_{i,1}, \ldots, y_{i,t})$
   (b) Set $\vec{z}_i \leftarrow \vec{m}_{0,i} + \vec{y}_i$
4. Set $\boldsymbol{Z} \leftarrow \mathsf{RowMatrix}(\ell, t, \vec{z}_1, \ldots, \vec{z}_\ell)$
5. For $i = 1, \ldots, \ell$:
   (a) Set $\boldsymbol{C}_i \leftarrow \mathsf{SingleRowMatrix}(\ell, t, i, \vec{c}_i)$
   (b) Set $\boldsymbol{D}_i \leftarrow \mathsf{Diag}(t, \vec{m}_{1,i} - \vec{m}_{0,i})$
```

```
 6. Define the $\mathbb{Z}_2$-linear function
        $f\colon (\{0,1\}^{\ell\times n})^{\ell} \to \{0,1\}^{\ell\times t}$
        $(\boldsymbol{X}_1,\dots,\boldsymbol{X}_\ell) \mapsto \left(\sum_{i=1}^{\ell}(-\boldsymbol{X}_i\boldsymbol{A}+\boldsymbol{C}_i)\cdot\boldsymbol{D}_i\right)+\boldsymbol{Z}$
 7. Compute $\hat{\mathsf{ct}} \xleftarrow{\$} \mathcal{LHE}.\mathsf{Eval\&Shrink}(\mathsf{pk}, f, \mathsf{ct}_1,\dots,\mathsf{ct}_\ell)$.
 8. For $i = 1,\dots,\ell$:
    (a) Set $\mathtt{DB}_i \leftarrow (y_{i,1}+(m_{1,i,1}-m_{0,i,1}),\dots,y_{i,t}+(m_{1,i,t}-m_{0,i,t}))$
    (b) For $j = 1,\dots,t$:
            Compute $\vec{r}_{i,j} \leftarrow \mathsf{PIR.Send}(\mathtt{DB}_i, \vec{q}_{i,j})$
 9. Set $\mathsf{ots}^\star \leftarrow (\{\mathsf{copir}_{S,i}\}_{i\in[\ell]}, \{r_{i,j}\}_{i\in[\ell],j\in[t]})$
10. Set $\mathsf{ots}^{\mathsf{dec}} \leftarrow \hat{\mathsf{ct}}$
11. Output $(\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}})$


dec-OTD: On input a batch subset $K \subseteq [k]$, a partial sender message $\mathsf{ots}'$ and an state $\mathsf{st}$:

 1. Parse $\mathsf{ots}$ as $\mathsf{ots} = (\mathsf{ots}^\star, \mathsf{ots}'_{\mathsf{dec}}) = ((\{\mathsf{copir}_{S,i}\}_{i\in[\ell]}, \{r_{i,j}\}_{i\in[\ell],j\in[t]}), \hat{\mathsf{ct}}')$
 2. Parse $\mathsf{st}$ as $\mathsf{st} = (\mathsf{sk}, \{\mathsf{st}_i, J_i\}_{i\in[\ell]}, \{\hat{\mathsf{st}}_{i,j}\}_{i\in[\ell],j\in[t]})$
 3. For $i = 1,\dots,\ell$ :
    (a) Compute $\overline{\boldsymbol{y}}_i \leftarrow (\overline{y}_{i,1},\dots,\overline{y}_{i,t}) \leftarrow \mathsf{coPIR.Retrieve}(\mathsf{copir}_{S,i}, \mathsf{st}_i)$
    (b) For $j = 1,\dots,t$:
            Compute $\overline{\boldsymbol{z}}_{i,j} \leftarrow \mathsf{PIR.Retrieve}(\mathsf{copir}_{S,i}, \hat{\mathsf{st}}_{i,j})$
    (c) Set $\vec{z}_i \leftarrow (z_{i,1},\dots,z_{i,t})$ where
            $z_{i,l} = \begin{cases} \overline{z}_{i,j} & \text{if } l = J_i[j] \\ \overline{y}_{i,\ell} & \text{otherwise} \end{cases}$ .
 4. Set $\boldsymbol{Z} \leftarrow \mathsf{RowMatrix}(\ell, t, \vec{z}_1,\dots,\vec{z}_\ell)$
 5. Set $S_K := \{(j \bmod \ell, j \text{ quo } \ell) \colon j \in K\} \subseteq [\ell]\times[t]$
 6. Compute $\hat{\boldsymbol{W}}' \leftarrow \mathcal{LHE}.\mathsf{pDecShrink}(K, \mathsf{sk}, \hat{\mathsf{ct}}')$, $\hat{\boldsymbol{W}}$ as
            $\hat{\boldsymbol{W}}[i,j] := \begin{cases} \hat{\boldsymbol{W}}'[i,j] & \text{if } (i,j)\in S_K \\ 0 & \text{if } (i,j)\in[\ell]\times[t]\setminus S_K \end{cases}$
    and $\boldsymbol{W} \leftarrow \hat{\boldsymbol{W}} - \boldsymbol{Z}$ .
 7. Let $\vec{w}_1,\dots,\vec{w}_\ell$ be the rows of $\boldsymbol{W}$.
 8. Output $(\vec{w}_i[j])_{(i,j)\in S_K} \in \{0,1\}^{|K|}$ .
```

Fig. 13: Decomposable Batch OT from Decomposable LHE, PIR, and co-PIR

Theorem 38 (Decomposable Batch OT from Decomposable LHE, PIR, and co-PIR). *Under the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim}, \mathsf{num}, \rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = \mathsf{dim}^c$ *(for any constant* $c > 1$*), and noise rate* $\rho = \mathsf{num}^{\varepsilon-1}$ *(for some constant* $\varepsilon < 1$*), if the following conditions are met:*

- *The batch size is of the form* $k = \ell \cdot t$*, where* $\ell, t \in \mathbb{N}$.
- $\mathcal{LHE}$ *is a rate-1 decomposable packed linearly homomorphic encryption scheme (definition 33) with plaintext space* $\{0,1\}^\ell$*, whose post-homomorphism shrunken ciphertexts have size* $k + \alpha(k)$*, where* $\alpha = o(1)$ *is some sublinear function;*
- coPIR *is a two-round co-PIR scheme over* $\{0,1\}$ *with poly-logarithmic communication complexity and parameterised by database size* t*,*
- PIR *is a two-round PIR scheme with poly-logarithmic communication complexity and sender privacy for database size* t*,*

then construction $\mathsf{dec\text{-}OT} = (\mathsf{dec\text{-}OTR}, \mathsf{dec\text{-}OTS}, \mathsf{dec\text{-}OTD})$ *from fig. 13 is a two-round decomposable batch OT with* $\alpha + \mathsf{polylog}$ *overhead.*

Proof.

- *Decomposable Correctness:* Observe that $\mathsf{dec\text{-}OTR}$ and $\mathsf{dec\text{-}OTS}$ are defined exactly as OTR and OTS in [BBDP22, Section 7], and furthermore that $\mathsf{dec\text{-}OTD}([k], \cdot, \cdot)$ is in fact the same algorithm (with the observation that, by decomposable correctness of $\mathcal{LHE}$, $\mathsf{DecShrink}(\cdot, \cdot) \equiv \mathsf{pDecShrink}([k], \cdot, \cdot)$) as $\mathsf{OTD}(\cdot, \cdot)$. Correctness therefore follows from [BBDP22, Theorem 1].

- *Receiver Security (Against Semi-Honest Sender):* Observe that $\mathsf{dec\text{-}OTR}$ is the same as the OTR from [BBDP22, Section 7], and therefore sender security follows from [BBDP22, Theorem 2].

- *Decomposable Sender Security (Against Semi-Honest Receiver):* Observe that $\mathsf{dec\text{-}OTS}$ is the same as the OTS from [BBDP22, Section 7], and that $\mathsf{dec\text{-}OTD}$ is only slightly modified from OTD. As such, the proof will closely follow that of [BBDP22, Theorem 3]. Let $\mathsf{Sim}^{\mathsf{dec}}_{\mathcal{LHE}}$ be a simulator as defined in the decomposable security of $\mathcal{LHE}$, and consider the following simulator $\mathsf{Sim}_{\mathsf{dec\text{-}OT}}$ which, on input $(1^\lambda, K, (m_i)_{i\in K}, \vec{b}, \mathsf{otr}, \mathsf{st})$, acts as follows:
  1. Parse otr as $\mathsf{otr} = (\mathsf{pk}, \boldsymbol{A}, \{\mathsf{ct}_i, \vec{c}_i, \mathsf{copir}_{R,i}\}_{i\in[\ell]}, \{q_{i,j}\}_{i\in[\ell], j\in[t]})$
  2. Parse st as $\mathsf{st} = (\mathsf{sk}, \{\mathsf{st}_i, J_i\}_{i\in[\ell]}, \{\hat{\mathsf{st}}_{i,j}\}_{i\in[\ell], j\in[t]})$
  3. For $i = 0, \dots, \ell t$:
     - If $i \in K$: Set $m_{b_i,i} \leftarrow m_i$ and $m_{1-b_i,i} \leftarrow 0$
     - If $i \notin K$: Set $m_{0,i} \leftarrow 0$ and $m_{1_i,i} \leftarrow 0$
  4. Set $\vec{m}_0 \leftarrow (m_{0,1}, \dots, m_{0,\ell t})$ and $\vec{m}_1 \leftarrow (m_{1,1}, \dots, m_{1,\ell t})$.
  5. Compute $(\mathsf{copir}_{S,i}, \vec{y}_i = (y_{i,1}, \dots, y_{i,m})) \xleftarrow{\$} \mathsf{coPIR.Send}(\mathsf{copir}_{R,i})$.
  6. For $i \in [\ell]$ and $j \in [t]$: Set $y'_{i,J_i[j]} \leftarrow y_{i,J_i[j]} + (m_{1,i,J_i[j]} - m_{0,i,J_i[j]})$, set $\overline{\mathsf{DB}}_{i,j} = (0, \dots, 0, y'_{i,J_i[j]}, 0, \dots, 0)$, and compute $\vec{r}_{i,j} \xleftarrow{\$} \mathsf{PIR.Send}(\mathsf{DB}_{i,j}, q_{i,j})$.
  7. Compute $\tilde{\mathsf{ct}} \xleftarrow{\$} \mathcal{LHE}.\mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, K, (m_s + z'_s)_{s\in K})$, where
$$z'_{j'} := \begin{cases} y'_{i\cdot\ell+j} & \text{if } j' = J_i[j] \text{ (where } (i,j) \in [\ell]\times[t]) \\ y_{i\cdot\ell+j'} & \text{otherwise} \end{cases}.$$
  8. Output $\mathsf{ots} = (\tilde{\mathsf{ct}}, \{\mathsf{copir}_{S,i}\}_{i\in[\ell]}, \{\vec{r}_{i,j}\}_{i\in[\ell], j\in[t]})$.

  The sequence of hybrids used to show indistinguishability between the real and the ideal worlds is the same as in the proof of [BBDP22, Theorem 3]:
  1. Start with the real experiment.
  2. For each $(i,j) \in [\ell]\times[t]$, by sender security of PIR we can set $\overline{\mathsf{DB}}_{i,j}$ to 0 everywhere except for $J_i[j]$.
  3. For each $i = 1, \dots, \ell$, by sender security of coPIR we can replace $y'_{i,J_i[j]} \leftarrow y_{i,J_i[j]} + (m_{1,i,J_i[j]} - m_{0,i,J_i[j]})$ (in such a way that $\overline{\mathsf{DB}}_{i,j} = (0, \dots, 0, y'_{i,J_i[j]}, 0, \dots, 0)$).
  4. We can replace $\hat{\mathsf{ct}}$ with $\tilde{\mathsf{ct}} \xleftarrow{\$} \mathsf{Sim}^{\mathsf{priv\text{-}dec}}_{\mathcal{LHE}}(1^\lambda, \mathsf{pk}, k, K, (m_i)_{i\in K})$ by applying lemma 34.

□

By combining lemmas 35 and 37 and theorem 38 we obtain corollary 39.

Corollary 39. *Assume the* QR *assumption and the binary LPN assumption* $\mathsf{LPN}(\mathsf{dim}, \mathsf{num}, \rho)$ *with dimension* $\mathsf{dim} = \mathsf{poly}(\lambda)$*, number of samples* $\mathsf{num} = \mathsf{dim}^c$ *(for any constant* $c > 1$*), and noise rate* $\rho = \mathsf{num}^{\varepsilon-1}$ *(for some constant* $\varepsilon < 1$*). Then for any* $\ell = \ell(\lambda)$*, there exists a decomposable two-round batch oblivious transfer for batch size* $k = \ell \cdot \mathsf{num}$ *where*

- *The receiver message* otr *has size* $(\ell^2 \cdot \mathsf{dim} + \ell \cdot \mathsf{num}^\varepsilon) \cdot \mathsf{poly}(\lambda) + k$
- *The sender message* $\mathsf{ots} = (\mathsf{ots}^\star, \mathsf{ots}^{\mathsf{dec}})$ *has size* $|\mathsf{ots}^\star| = (\mathsf{num} + \ell \cdot \mathsf{num}^\varepsilon) \cdot \mathsf{poly}(\lambda)$ *and* $|\mathsf{ots}^{\mathsf{dec}}| = k$*.*

In particular, for appropriate parameters (sufficiently large ℓ*, and* num *sufficiently larger than* ℓ*),* $|\mathsf{otr}| = k + o(k)$*, and* $|\mathsf{ots}^\star| = o(k)$*.*