Fast White-Box Adversarial Streaming Without a Random Oracle

Abstract. We study problems in the white-box adversarially robust streaming model. Streaming algorithms work with a large stream of data and aim to respond to queries regarding desired statistics of the data using limited memory and fast per item processing time, i.e., update time. Recently, the question of adversarially robust streaming, where the stream is allowed to depend on the randomness of the streaming algorithm, has gained a lot of attention. In this work, we consider a strong white-box adversarial model (Ajtai et al. PODS 2022), in which the adversary has access to all past random coins and the parameters used by the streaming algorithm. We also consider and define a related white-box adversarially robust distributed model. We note that both notions are also related to property preserving hashing in the cryptography literature. We consider the sparse recovery problem, which has been used for tasks such as distinct element estimation, low rank approximation of matrices and tensors, and finding a maximum matching. The main drawback of all previous work is that it requires a *random oracle*, which is especially problematic in the streaming model since the amount of randomness is counted in the space complexity of a streaming algorithm. Another issue with previous constructions is that many of them suffer from either large update time or rely on non-standard cryptographic assumptions. We construct a near-optimal solution for the sparse recovery problem in white-box adversarial streams, based on the more standard subexponentially secure Learning with Errors assumption. Importantly, our solution is the first without a random oracle and with polylogarithmic per item processing time. We also give related results in the distributed model. Our constructions are based on homomorphic encryption schemes satisfying very mild structural properties that are currently satisfied by most known schemes.

1 Introduction

The streaming model of computation has emerged as an increasingly popular paradigm for the analysis of massive datasets, where the overwhelming size of the data places restrictions on the memory, computation time, and other resources available to the algorithm. In the streaming model, the input data is implicitly defined through a stream of data elements that sequentially arrive one-by-one. One can also delete a previous occurrence of an item. An algorithm makes one pass over the stream and uses limited space to approximate a desired function of the input. In the formalization of Alon, Matias, and Szegedy [5], the data is represented as an underlying n-dimensional vector that is initialized to 0^n, and then undergoes a sequence of additive updates to its coordinates. The algorithm

aims to optimize the space and time to process these updates, while being able to respond to queries about this underlying vector.

This model captures key resource requirements of algorithms for many practical settings such as Internet routers and traffic logs [51,74,66,70,21,22], databases [62,25,29,52,64,16], sensor networks [40,31], financial transaction data [3], and scientific data streams [20,67]. See [57] for an overview of streaming algorithms and their applications.

A large body of work on streaming algorithms has been designed for *oblivious* streams, for which the sequence of updates may be chosen adversarially, but it is chosen independently of the randomness of the streaming algorithm. Recently, there is a growing body of work on robust streaming algorithms in the black-box adversarial model [9,8,37,73,4,48,15,56,6,7,17], in which the adversary can monitor the outputs of the streaming algorithm and choose future stream updates based on these outputs. While useful in a number of applications, there are other settings where the adversary may also have access to the internal state of the algorithm, and this necessitates looking at a stronger adversarial model known as a white-box adversarial stream.

1.1 White-box Adversarial Streaming Model

We consider the white-box adversarial streaming model introduced in [2], where a sequence of stream updates is chosen adaptively by an adversary who sees the full internal state of the algorithm at all times, including the parameters and the previous randomness used by the algorithm. The interaction between an adversary and the streaming algorithm is informally depicted as the following game.

Consider a multi-round, two-player game between StreamAlg*, the streaming algorithm, and an adversary $\mathcal{A}$. Prior to the beginning of the game, fix a query function f, which asks for a function of an underlying dataset that will be implicitly defined by the stream. In each round:*

1. *$\mathcal{A}$ computes an update x for the stream, which depends on all previous stream updates, all previous internal states, and the randomness used by* StreamAlg*.*
2. StreamAlg *acquires a fresh batch of random bits, uses x to update its data structures, and (if asked) outputs a response to the query function f.*
3. *$\mathcal{A}$ observes the random bits, the internal state of* StreamAlg*, and the response.*

The goal of $\mathcal{A}$ is to make StreamAlg *output an incorrect query at some round during the game.*

We give a more formal Definition 2.12 in Section 2.3, but give a few motivating applications here.

The white-box adversarial model captures characteristics of many real-world attacks, where an adaptive adversary has access to the entirety of the internal states of the system. In comparison to the oblivious stream model or the black-box adversarial model [8], this model allows for richer adversarial scenarios.

For example, consider a distributed setting where a centralized server collects statistics of a database generated by remote users. The server may send components of its internal state S to the remote users in order to optimize the total communication over the network. The remote users may use S in some process that generates downstream data. Thus, future inputs depend on the internal state S of the algorithm run by the central coordinator. In such settings, the white-box robustness of the algorithms is crucial for optimal selection of query plans [62], online analytical processing [64], data integration [16], and data warehousing [25].

Many persistent data structures provide the ability to quickly access previous versions of information stored in a repository shared across multiple collaborators. The internal persistent data structures used to provide version control may be accessible and thus visible to all users of the repository. These users may then update the persistent data structure in a manner that is not independent of previous states [26,28,47].

Dynamic algorithms often consider an adaptive adversary who generates the updates upon seeing the entire data structure maintained by the algorithm during the execution [18,19,60]. For example, [71] assumes the entire state of the algorithm (including the set of randomness) is available to the adversary after each update, i.e., a white-box model.

Moreover, robust algorithms and adversarial attacks are important topics in machine learning [68,36], with a large body of recent literature focusing on adversarial robustness of machine learning models against white-box attacks [41,55,61,69,24,50,53].

1.2 Prior Work in the White-Box Model

We aim to design *white-box adversarially robust* (WAR) streaming algorithms with provable correctness guarantees against such powerful adversaries. In general, it is difficult to design non-trivial WAR algorithms, due to the fact that many widely used space-saving techniques are subject to attacks when all parameters and generated randomness of the system are immediately revealed to the adversary In the streaming community, both [2] and [27] explicitly study the white-box model for data streams and suggest the use of cryptography in designing WAR algorithms (see also [8] for the use of cryptography for black box robustness). A similar model to the white-box model has also been independently developed in the cryptography community. In property preserving hashing [11] one requires that the hash h allows, given $h(\boldsymbol{x}), h(\boldsymbol{y})$ for two inputs $\boldsymbol{x}, \boldsymbol{y}$, to compute a property $P(\boldsymbol{x}, \boldsymbol{y})$ such as distances of $\boldsymbol{x}$ and $\boldsymbol{y}$ in various norms. The robust versions, similar in spirit to white-box adversarially robust streaming, require that it is computationally hard to find $\boldsymbol{x}, \boldsymbol{y}$ so that from the hashes, one cannot predict the property well. There is some overlap between the properties studied in this model and the problems studied in the white-box model, such as estimating the number of non-zero entries of a vector. We compare the models more below.

A notable drawback of the solutions proposed in both communities is their reliance on either a *random oracle* or a prohibitively long random string. That is, a key component to their algorithms is a random matrix $\mathbf{A}$ for which it is hard to find a vector $\boldsymbol{x}$ with small entries for which $\mathbf{A}\boldsymbol{x}$ is sparse or even zero. Unfortunately, it is unknown how to generate $\mathbf{A}$ pseduorandomly and still maintain white-box adversarial robustness. This is especially problematic in data streams, where the resource measure is the total space complexity, including the randomness, and there is a large body of work on derandomizing such algorithms, see, e.g., [58,42,46,45]. This motivates a central question of this work:

Question: Do there exist efficient streaming algorithms in the white-box model without a random oracle?

Another drawback of prior work is the time required of such schemes. The sketching matrices $\mathbf{A}$ used in [2,27] are random dense matrices, and thus the time to process each stream item, which we refer to as the *update time*, is as large as the number of rows of $\mathbf{A}$, which can be large depending on the application. For some problems, such as estimating the number of non-zeros of an underlying vector $\boldsymbol{x}$ in a stream up to a constant factor, it was shown that the matrix $\mathbf{A}$ can be sparse [11]. However, the results of [11] are based on the less studied Sparse Short Vector assumption, require a random oracle or storing the entire matrix $\mathbf{A}$, and do not apply to the main problem of sparse recovery that we study. However, by replacing the list-decoding sketch in the followup work [39] with a sparse compressed sensing scheme based on expanders [10], it seems possible to significantly reduce the update time even for the central problem of sparse recovery that we study. Unfortunately, [39] also only seems to work in the random oracle model or require the storage of a very long random string. Thus, achieving fast update time while accounting for the total memory including that to store randomness, is an essential goal.

We now define the central problem we consider in the white-box model, which is the k-sparse recovery problem as follows:

Sparse Recovery: given an input vector $\boldsymbol{x}$, if $\boldsymbol{x}$ contains at most k non-zero entries, recover $\boldsymbol{x}$. Otherwise, report *invalid input*.

A primitive for sparse recovery in the white-box model can be used for tasks such as distinct element estimation, low rank approximation of matrices and tensors, and finding a maximum matching [2,27]. Sparse recovery is also itself a critical data acquisition and processing problem that arises in signal and image processing, machine learning, data networking, and medicine, see [2,27] and the references therein.

We note that a weaker definition of sparse recovery assumes that the input vector $\boldsymbol{x}$ is promised to be k-sparse, and only under this promise recovers $\boldsymbol{x}$. If the input is not k-sparse, the algorithm can output any vector. This problem can in fact be solved deterministically (and thus also in the white-box model) with low memory (see, e.g., [10], for a version with fast update time, though other simpler schemes based on MDS codes exist). The drawback under this weaker definition

though is that, without knowing the sparsity of the input ahead of time, a user of the algorithm cannot tell whether the output is a correct recovery or garbage. In the latter case, the client might ignorantly use the garbage output in subsequent applications and propagate such an error.

As discussed above, the main issues with previous work on sparse recovery in the white-box setting [2,27] or related property preserving hashing works [11,39] is that they rely on a random oracle, or do not apply to the sparse recovery problem itself, or do not have fast update time.

1.3 Our Results

We let n be the dimension of the underlying input vector, and for notational convenience, we assume the stream length is at most a polynomial in n.

We also assume for notational convenience that the entries of the vector are integers bounded in absolute value by $\mathsf{poly}(n)$ at all times during the stream. We use $\widetilde{\mathcal{O}}(f)$ to denote $f \cdot \mathsf{poly}(\log n)$. We denote the security parameter of our cryptographic assumption as λ. Our robustness guarantees hold against adversaries that run in time polynomial in n. In our applications, the efficiency metrics will depend on the security parameter λ used for the cryptographic components. We can set $\lambda = n^c$ for an arbitrary small constant $c > 0$. Further, assuming subexponential time assumptions we could also set λ to be a large enough polylogarithmic function of n. Let $\ell(\lambda)$ denote a fixed polynomial in λ that represents the maximum run-time of various cryptographic algorithms involved. By setting $\lambda = n^c$ appropriately for a small enough constant c we can always set $\ell(\lambda)$ to be a small enough polynomial in n (or a polylogarithmic function of n assuming subexponential time assumptions). Our results are described in terms of $\ell(\lambda)$.

Our main result is informally stated as follows:

Theorem 1.1 (Informal). *Assuming the hardness of the Learning with Errors (LWE) problem, there is a WAR streaming algorithm for k-sparse recovery, which:*

- *takes $\widetilde{\mathcal{O}}(k + \ell(\lambda))$ bits of space,*
- *has $\widetilde{\mathcal{O}}(1 + \ell(\lambda))$ update time, and*
- *has $\widetilde{\mathcal{O}}(k \cdot (k^c + \ell(\lambda)))$ report time for an arbitrarily small constant $c > 0$.*

This matches (up to a poly-logarithmic factor) the optimal space and update time complexity for the weaker k-sparse recovery problem under the oblivious streaming setting. On the other hand, assuming polynomial security of LWE, there is a multiplicative overhead of n^ϵ for arbitrary constant $\epsilon > 0$. This matches the complexity of the previous work [27] that was proven to be secure in the random oracle model.

We also study the sparse recovery problem in a white-box distributed model, which is a computational framework that is closely related to the data stream model. See Section A.1 for a formal discussion of the model, which involves multiple servers who communicate with a so-called coordinator. A WAR streaming

algorithm can be naïvely converted to a WAR distributed algorithm under certain conditions, yet the resulting processing time of the servers is $\mathcal{O}(n)$ times that of the streaming update time. We show that we can achieve a near-optimal processing time in a white-box distributed model assuming the *polynomial* hardness of Ring-LWE. This is in contrast to the suboptimal time implied by the naïve conversion above and we also do not require a subexponential hardness assumption. This is due to the fact that Ring-LWE supports SIMD (Single-Instruction Multiple-Data) style operations. We informally state our result as follows:

Theorem 1.2 (Informal). *Assuming the hardness of Ring-LWE, there exists a WAR distributed protocol for k-sparse recovery, which:*

- *uses $\widetilde{\mathcal{O}}(k + \ell(\lambda))$ bits of communication between each server and the coordinator,*
- *has $\widetilde{\mathcal{O}}(n)$ processing time on each server, and*
- *has the coordinator spend $\widetilde{\mathcal{O}}(\max(n, k^{1+c}))$ time to output the solution given the messages from the servers, for an arbitrarily small constant $c > 0$.*

We summarize the complexities of our algorithms in Table 1, as compared to the best-known upper bounds for these problems in the white-box adversarial model.

Table 1. A summary of the bit complexities and runtime of our k-sparse recovery algorithm, as compared to the best-known upper bounds for these problems in the white-box adversarial model. The TIME columns refer to the update time in the streaming model and processing time in the distributed model. c denotes an arbitrarily small positive constant.

	Model	Assumption	Space	Time	Rand. Oracle?
Prev.	Streaming	subexp SIS	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(k)$	✓
	Distributed	subexp SIS	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(nk)$	✓
Ours	Streaming	subexp LWE	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(1)$	✗
	Streaming	poly LWE	$\widetilde{\mathcal{O}}(n^c + k)$	$\widetilde{\mathcal{O}}(n^c)$	✗
	Distributed	subexp LWE	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(n)$	✗
	Distributed	poly LWE	$\widetilde{\mathcal{O}}(n^c + k)$	$\widetilde{\mathcal{O}}(n^{1+c})$	✗
	Distributed	poly Ring-LWE	$\widetilde{\mathcal{O}}(n^c + k)$	$\widetilde{\mathcal{O}}(n)$	✗

Similar to [27], generalizing our construction produces low-rank matrix and tensor recovery algorithms, again with the crucial property that they do not assume a random oracle or store a large random seed.

Consider a data stream updating the entries of an underlying matrix or tensor, which can be represented as its vectorization in a stream. Let $n \geq m$ be the dimensions of an underlying input matrix, and let n be the dimension

parameter of an underlying data tensor in $\mathbb{R}^{n^d}$ for some $d \in \mathcal{O}(1)$ (we assume all dimensions of the tensor are the same only for ease of notation). We have the following results in the streaming model:

Theorem 1.3 (Informal). *Assuming the hardness of the Learning with Errors (LWE) problem, there is a WAR streaming algorithm for rank-k matrix recovery, which takes $\widetilde{\mathcal{O}}(nk+\ell(\lambda))$ bits of space. There is also a WAR streaming algorithm for rank-k tensor recovery, which takes $\widetilde{\mathcal{O}}(nk^{\lceil \log d \rceil} + \ell(\lambda))$ bits of space.*

We remark that the above matrix recovery algorithm performs $\widetilde{\mathcal{O}}(nk)$ measurements using n-sparse matrices (plus a relatively small FHE hash) which can be converted into a distributed protocol with $\widetilde{\mathcal{O}}(n^2k)$ processing time. This improves upon the matrix recovery algorithm in the previous work [27] which, if implemented in a distributed setting, would require $\widetilde{\mathcal{O}}(n^3k)$ processing time at each server.

Additional Applications. Our k-sparse recovery algorithm can be used to construct an L_0-norm estimation algorithm following the same recipe in [27]. In short, by running our k-sparse recovery algorithm with the sparsity parameter k set to n^{1-w} for some constant w, we can get an n^w-approximation to the L_0 norm of a vector. In comparison to [27], our construction again removes the need for a random oracle or the need to store a lot of randomness.

In addition, our construction implies a family of Homomorphic Collision-Resistance hash functions defined in [39], with concrete efficiency, security against white-box adversaries (i.e., adversaries with access to the randomness of sampling the hash function), and especially short hash function descriptions, which may be of independent interest.

1.4 Technical Overview

We leverage structural properties of lattice-based assumption to address all drawbacks listed above. In particular, departing from the previous construction of [27], we work with a suitable FHE scheme to achieve the desired property. We start with an intuitive explanation of the previous construction in [27].

Previous Construction One general framework for solving the sparse recovery problem is to run a weak recovery algorithm as a black-box subroutine, which potentially provides garbage output, in addition to an additional "tester" scheme to verify whether the recovery output matches the original input (i.e., is a correct recovery) or is garbage. In the oblivious streaming setting, such a "tester" can be implemented using a space and time-efficient randomized sampler [23]. However, this approach is not WAR as the randomness of the sampler is revealed and thus not independent of the stream.

Instead, [27] uses a random matrix $\mathbf{A}$ to implement a WAR tester, which is assumed to be heuristically compressed by an idealized hash function. During the stream, [27] maintains a hash $h = \mathbf{A}\boldsymbol{x}$, where $\boldsymbol{x}$ is the data vector implicitly

defined by the stream. This is easy to maintain since $\mathbf{A}$ is linear and the stream undergoes additive updates of the form $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \Delta$ to the coordinates $\boldsymbol{x}_i$ of $\boldsymbol{x}$. In the testing stage, it acknowledges an output $\boldsymbol{x'}$ to be a correct recovery if and only if $\boldsymbol{x'}$ is k-sparse and $\mathbf{A}\boldsymbol{x'}$ equals the hash h. The idea is that if there exists a k-sparse collision $\boldsymbol{x'}$ satisfying $\mathbf{A}(\boldsymbol{x} - \boldsymbol{x'}) = 0$ and $\boldsymbol{x'} \neq \boldsymbol{x}$, then an adversary can brute-force over all k-sparse vectors to find such $\boldsymbol{x} - \boldsymbol{x'}$, which is a solution to the SIS problem. To spawn n matrix columns and support the brute-force reduction, the security parameter (i.e., the number of rows of $\mathbf{A}$) needs to be in $\omega(k \log n)$, assuming the exponential security of SIS.

Efficient Reduction. The number of matrix rows in the previous construction depends on the sparsity parameter k due to a brute-forcing step in the reduction. To grant enough time for iterating through all k-sparse vectors (with $\mathsf{poly}(n)$-bounded entries), the security parameter has to be greater than $k \log n$, which results in $\widetilde{\mathcal{O}}(k)$ time to process every single update. Our first idea is to construct a poly-time reduction in order to remove such a dependency on k. This allows us to achieve better space and time efficiency, in addition to basing the construction on a polynomial hardness assumption.

The observation is that the non-existence of a sparse collision is an overly strong condition. Instead, for our purpose, we only need to ensure that such a collision, if it exists, cannot be found by the poly-time weak recovery scheme that the algorithm runs. Thus, in the reduction, instead of brute-forcing over all k-sparse vectors, we let the adversary run the same poly-time weak recovery scheme that the algorithm does. If a collision $\boldsymbol{x'}$ satisfying $\mathbf{A}\boldsymbol{x} = \mathbf{A}\boldsymbol{x'}$ is output by the scheme, then the adversary can still break the security assumption.

Reducing the Amount of Randomness. The rest of our effort will be attributed to reducing the size of the state of the algorithm. Rather than using a truly random giant matrix to hash the data as in the previous construction, we would like to design a short digest (ideally linear-sized in the security parameter) and generate the columns of the hash matrix on the fly. This naturally yields correlated columns as opposed to the truly random columns in an SIS matrix. We aim to show that in this setting, the chance of finding a collision is still negligible.

Streaming Friendly Collision-Resistant Hash Our main insight from cryptography to address the issue above is that we could construct a hash function family $\mathcal{H}$, with special properties to help solve our streaming problem:

- The hash key $h \leftarrow \mathcal{H}$ is small in size, ideally polynomial in the security parameter (or even polylogarithmic assuming subexponential time assumptions), in particular independent of the dimension n of the stream vector.
- The hash should be computable in a streaming fashion and it should be updatable where each update takes polynomial time in λ (or even polylogarithmic time assuming subexponential time assumptions) independent of n or k or other parameters.

– The hash key should be pseudorandom. In other words, in any real world implementation in the streaming setting the model does not allow us to use any structured randomness (such as with hidden planted secrets). In the proof however, we could introduce these secrets which would be important for our construction.
– The hash should satisfy collision-resistance.

Constructing Streaming Friendly Hashing from FHE. How could we build such a hash function? The SIS based hash function does have a fast update time. On the other hand, we could take an arbitrary collision resistant function that compresses the number of bits by a factor of $1/2$ and use it to perform a Merkle style hashing to hash strings of arbitrary length. Such a hash will have $\mathsf{poly}(\lambda)$ size hashing keys independent of the stream length. Unfortunately, the first idea suffers from a large hashing key size, which could be shortened in the random order (RO) model by using RO to store the SIS matrix. Moreover, the second proposal will not support positive and negative stream updates to the coordinates of the underlying vector.

Our main idea is a modular construction that leverages a suitably chosen FHE scheme satisfying mild structural properties (that are satisfied currently by most schemes based on LWE and Ring-LWE) to build such a hash function. Our ideas are somewhat inspired from [14], which constructed ABE with shorter public keys to encrypt with respect to attributes of unbounded length.

We directly explain the idea in the context of our construction.

In our hash function, the hash key consists of $L = O(\log n)$ ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_L$. We assume that our FHE scheme has pseudorandom ciphertexts and keys. In our actual scheme, these ciphertexts will be generated as random strings. It is only in the proof that they will correspond to encryptions of carefully chosen values.

The idea is that from these L ciphertexts, we will derive n evaluated ciphertexts $\widehat{\mathsf{ct}}_j$ for $j \in [n]$ by evaluating L ciphertexts on simple functions $C_1, \ldots, C_n$ (described later). These n ciphertexts will correspond to columns of the SIS type hashing. The hash of a vector $\boldsymbol{x}$ is computed as the linear combination $\sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}_i$ where $\boldsymbol{x}_i$ is the i^{th}coordinate. Here the sum is done over the field on which the FHE ciphertexts live.

The circuits $C_1, \ldots, C_n$ are chosen keeping the security proof in mind. The ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_L$ are "thought" to encrypt a value $m \in \{0,1\}^L$ that will be chosen in the proof, and each $C_i(\cdot)$ on input m outputs 1 if $i = m$ and 0 otherwise. Observe that these circuits are polynomial in $L = O(\log n)$ sized and therefore can be evaluated on $\mathsf{ct}_1, \ldots, \mathsf{ct}_L$ in polynomial in $(\lambda, \log n)$ time guaranteeing fast updates.

How do we prove security? The idea is that if an adversary breaks collision-resistance it will produce two vectors $\boldsymbol{x} \neq \boldsymbol{x'}$, so that the corresponding hashes $\sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}_i = \sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}'_i$. Suppose one could know exactly one coordinate $v \in [n]$ so that $\boldsymbol{x}_v \neq \boldsymbol{x}'_v$. Then we could set $m = v$. In this case $\widehat{\mathsf{ct}_v}$ encrypts 1 and the other ciphertexts encrypt 0. In this case, $\sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}_i = \sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}'_i$

would lead to a contradiction if our FHE scheme has a special property (that is satisfied by common FHE schemes). The property is that if $\widehat{\mathsf{ct}_i}$ encrypts bit $\mu_i \in \{0,1\}$ and $\boldsymbol{x}_i$ are small norm then there is a decryption algorithm that uniquely binds $\sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}_i$ to $\sum_{i\in[n]} \mu_i \boldsymbol{x}_i$. If this holds, then the equality $\sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}_i = \sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}'_i$ would imply $\boldsymbol{x}_v = \boldsymbol{x}'_v$ which is a contradiction.

This leads us to two remaining problems:

1. Given that we do not know $\boldsymbol{x}$ and $\boldsymbol{x'}$ ahead of time, how do we decide which index m to encrypt such that it will help identify whether $\boldsymbol{x} = \boldsymbol{x'}$?
2. Moreover, a WAR algorithm cannot actually encrypt FHE ciphertexts, since all randomness and parameters used by the algorithm will be exposed to the adversary. Instead, we want our digest to be unstructured and truly random.

FHE with a Pseudorandom Property. We solve both issues above by considering a pseudorandom property of the FHE scheme and show that a random guess of the index m actually suffices. We assume that the distributions of public keys and ciphertexts of the FHE scheme are indistinguishable, respectively, from some truly random distributions from the perspective of the adversary.

We sample truly random digests in our construction, and perform a random guess of $m \in [n]$ in the proof. If an adversary finds a collision pair $\boldsymbol{x} \neq \boldsymbol{x'}$ with probability γ, then with probability $p \geq \gamma/n$, our guess pins down a coordinate of disagreement $\boldsymbol{x}_m \neq \boldsymbol{x'}_m$. In the proof, we then switch the ciphertexts to encrypt m. The chance of guessing the disagreement is still p, as otherwise one can distinguish the pseudorandom ciphertext. However, as argued earlier, $\boldsymbol{x}_m \neq \boldsymbol{x'}_m$ roughly implies $\sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}_i \neq \sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}'_i$, and thus two distinct vectors cannot form a hash collision that fools our "tester".

In our complexity analysis, we instantiate the FHE scheme using GSW, which satisfies the additional properties required by our construction. We remark that in addition to GSW [33], most current FHE schemes based on LWE and Ring-LWE also exhibit these properties, and thus they are not an artifact obtained from a particular FHE scheme.

Ring-LWE for the Distributed Setting. In the distributed model, each remote server receives as input a length-n vector. The goal is to design a protocol such that each server can efficiently compute a short summary of their partition of data. Adapting the above streaming algorithm to this setting, each server needs to evaluate n ciphertexts in order to compute a linear combination $\sum_{i\in[n]} \widehat{\mathsf{ct}_i}\boldsymbol{x}_i$.

Using a polynomially secure LWE-based FHE scheme, it is unclear how to speed up such n-many individual evaluations as each computation takes $\mathsf{poly}(\lambda)$ operations, resulting in total complexity of $n\,\mathsf{poly}(\lambda)$ where λ is the security parameter. However, appealing to the SIMD property of Ring-LWE and packing the data as ring elements, we can represent the input vectors as partitioned segments of length proportional to the security parameter (rather than n individual coordinates), and efficiently operate on them segment-by-segment using the Fast Fourier Transform. This improves the processing time of each server to be almost linear in the dimension of the data. We refer the reader to Section A.1 for more details.

Summary. Leveraging FHE or lattice-based assumptions in algorithm design are relatively new techniques. We think it would be good to explore their applications more broadly in robust algorithms, beyond the recovery problems or even the adversarial settings considered here.

2 Preliminaries

Basic Notations. Let $\mathbb{1}(p)$ for some proposition p be an indicator variable, which equals 1 if p is true and 0 otherwise. Define $[a, b] := \{a, a+1, \ldots, b\}$, $[n] := [1, n]$. For a finite set S, we write $x \xleftarrow{\$} S$ to denote uniformly sampling x from S. Our logarithms are in base 2. For $n \in \mathbb{R}$, we use $\lceil n \rceil$ to denote rounding n up to the nearest integer. For a vector $\boldsymbol{x}$, $\|\boldsymbol{x}\|_0$ denotes the ℓ_0 norm of $\boldsymbol{x}$, which is the number of its non-zero entries. We abbreviate PPT for probabilistic polynomial-time. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is negligible if for every constant $c > 0$ there exists an integer N_c such that $\mathsf{negl}(x) < x^{-c}$ for all $x > N_c$.

2.1 Lattice Preliminaries

General definitions. A *lattice* $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^g$, or equivalently the set $\mathcal{L}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_m) = \{\sum_{i=1}^{g} x_i \boldsymbol{b}_i \ : \ x_i \in \mathbb{Z}\}$ of all integer combinations of g vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_m \in \mathbb{R}^g$ that are linearly independent. Such $\boldsymbol{b}_i$'s form a *basis* of $\mathcal{L}$. The lattice $\mathcal{L}$ is said to be *full-rank* if $g = m$. We denote by $\lambda_1(\mathcal{L})$ the so-called first minimum of $\mathcal{L}$, defined to be the length of a shortest non-zero vector of $\mathcal{L}$.

Discrete Gaussian and Related Distributions For any $s > 0$, define $\rho_s(\boldsymbol{x}) = \exp(-\pi\|\boldsymbol{x}\|^2/s^2)$ for all $\boldsymbol{x} \in \mathbb{R}^g$. We write ρ for ρ_1. For a discrete set S, we extend ρ to sets by $\rho_s(S) = \sum_{\boldsymbol{x} \in S} \rho_s(\boldsymbol{x})$. Given a lattice $\mathcal{L}$, the *discrete Gaussian* $\mathcal{G}_{\mathcal{L},s}$ is the distribution over $\mathcal{L}$ such that the probability of a vector $\boldsymbol{y} \in \mathcal{L}$ is proportional to $\rho_s(\boldsymbol{y})$: $\Pr_{X \leftarrow \mathcal{G}_{\mathcal{L},s}}[X = \boldsymbol{y}] = \frac{\rho_s(\boldsymbol{y})}{\rho_s(\mathcal{L})}$.

The discrete Gaussian is not bounded but with overwhelming probability a sample from $\mathcal{G}_{\mathcal{L},s}$ is bounded by $s\sqrt{g}$ in ℓ_2 norm.

Lattice problems. We consider worst-case and average-case problems over lattices. We start by defining the gap version of the shortest vector problem.

Definition 2.1. *For any function $\gamma = \gamma(g) \geq 1$, the decision problem GapSVP_γ (Gap Shortest Vector Problem) is defined as follows: the input is a basis $\boldsymbol{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^g$ and a real number $d > 0$ that satisfy the promise that $\lambda_1(\mathcal{L}(\boldsymbol{B})) \leq d$, or $\lambda_1(\mathcal{L}(\boldsymbol{B})) \geq \gamma d$. The goal is to decide whether $\lambda_1(\mathcal{L}(\boldsymbol{B})) \leq d$, or $\lambda_1(\mathcal{L}(\boldsymbol{B})) \geq \gamma d$.*

This problem is known to be NP hard for $\gamma = g^{o(1)}$ [49,38], and is known to be inside $\mathsf{NP} \cap \mathsf{coNP}$ for $\gamma = O(\sqrt{g})$ [1,35]. The problem is believed to be intractable by polynomial time adversaries even for $\gamma = 2^{g^\epsilon}$ for $0 < \epsilon < 1$.

Next, we define the Learning with Errors problem LWE.

Definition 2.2 (Learning with Errors). *For any integer dimension $g \in \mathbb{N}$, modulus $q \in \mathbb{N}$, sample complexity $h \geq \Omega(g \cdot \log q)$ and a parameter $\sigma \in \mathbb{R}^{\geq 0}$. Define an error distribution as the discrete Gaussian $\chi_\sigma^h := \mathcal{G}_{\sigma,\mathbb{Z}^h}$. We define $\mathsf{LWE}_{g,h,q,\chi_\sigma^h}$ as follows: Sample a matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{g \times h}$ and a secret $\boldsymbol{s} \in \mathbb{Z}_q^{1 \times g}$. Sample an error vector $\boldsymbol{e} \leftarrow \chi_\sigma^h$. The problem is to distinguish $(\mathbf{A}, \boldsymbol{s}\mathbf{A}+\boldsymbol{e} \mod q)$ from $(\mathbf{A}, \boldsymbol{r})$ where $\boldsymbol{r}$ is a random in $\mathbb{Z}_q^{1 \times h}$.*

Under quantum and classical reductions it can be shown that $\mathsf{LWE}_{g,h,q,\chi_\sigma}$ is harder to solve than $\mathsf{Gap\text{-}SVP}_\gamma$ with $\gamma = g\frac{q}{\sigma}$.

We also consider the Ring-LWE problem, which is an average-case lattice problems over ideal lattices.

Definition 2.3 (Ring-LWE). *Let $q \geq 1$ be a prime number, and let g be a power of 2. Let g, σ be positive integers. The $\mathsf{Ring\text{-}LWE}_{g,h,q,\chi_\sigma}$ problem is defined as follows: Consider $R = \frac{\mathbb{Z}_q[x]}{x^g+1}$. Sample at random $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_h \leftarrow R$ and a secret polynomial $\boldsymbol{s} \leftarrow R$. Sample h error polynomials $\boldsymbol{e}_1, \cdots, \boldsymbol{e}_h$ such that each $\boldsymbol{e}_i$ is a g-dimensional vector $\boldsymbol{e}_i \leftarrow \chi_\sigma = \mathcal{G}_{\sigma,\mathbb{Z}^g}$. The problem is to distinguish the tuple $(\boldsymbol{a}_1, \cdots, \boldsymbol{a}_h, \{\boldsymbol{s}\boldsymbol{a}_i + \boldsymbol{e}_i\}_{i\in[h]})$ from $(\boldsymbol{a}_1, \cdots, \boldsymbol{a}_h, \{\boldsymbol{r}_i\}_{i\in[h]})$ where $\boldsymbol{r}_1, \cdots, \boldsymbol{r}_h$ are random polynomials in R.*

The Ring-LWE problem is as hard as worst-case problems over ideal lattices [54]. Ring LWE is sometimes advantageous to use over LWE because of shorter parameters. One ring LWE element is of dimension g, and requires only one ring element as a coefficient, as opposed to g vectors in the case of LWE. Moreover, ring multiplication can be sped up using number-theoretic transforms [65].

2.2 Fully Homomorphic Encryption

We review the definition of fully homomorphic encryption.

Notation. We denote the security parameter of a cryptographic scheme by λ. We write PPT_λ to denote probabilistic poly-time in λ. We say two ensembles of distributions $\{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{\mathcal{D}'_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable, denoted by $\approx_c$, if for any non-uniform PPT adversary $\mathcal{A}$ there exists a negligible function negl such that $\mathcal{A}$ can distinguish between the two distributions with probability at most $\mathsf{negl}(\lambda)$.

Remark 2.4. We also define a $(\mathcal{T}, \epsilon)$-*indistinguishability*, where $\mathcal{T}$ and ϵ are two functions in λ. We say that two distribution ensembles $\{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{\mathcal{D}'_\lambda\}_{\lambda\in\mathbb{N}}$ are $(\mathcal{T}, \epsilon)$-indistinguishable, denoted by $\approx_{(\mathcal{T},\epsilon)}$, if there exists an integer λ_0 such that for all $\lambda > \lambda_0$, for every adversary $\mathcal{A}$, $\mathcal{A}$ can distinguish between the two distributions with probability at most $\epsilon(\lambda)$.

Definition 2.5 (FHE). *A fully homomorphic encryption FHE scheme is a tuple of PPT_λ algorithms $\mathsf{FHE} = (\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$ with the following properties:*

- $\mathsf{FHE.Setup}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$ *: On input a security parameter* λ*, the setup algorithm outputs a key pair* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{FHE.Enc}(\mathsf{pk}, \mu) \to \mathsf{ct}$ *: On input a public key* pk *and a message* $\mu \in \{0,1\}$*, the encryption algorithm outputs a ciphertext* ct.
- $\mathsf{FHE.Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \cdots, \mathsf{ct}_\ell) \to \widehat{\mathsf{ct}}$ *: On input a public key* pk*, a circuit* $C : \{0,1\}^\ell \to \{0,1\}$ *of depth at most* $\mathsf{poly}(\lambda)$*, and a tuple of ciphertexts* $\mathsf{ct}_1, \cdots, \mathsf{ct}_\ell$*, the evaluation algorithm outputs an evaluated ciphertext* $\widehat{ct}$.
- $\mathsf{FHE.Dec}(\mathsf{pk}, \mathsf{sk}, \widehat{\mathsf{ct}}) \to \widehat{\mu}$ *: On input a public key* pk*, a secret key* sk*, and a ciphertext* $\widehat{\mathsf{ct}}$ *, the decryption algorithm outputs a message* $\widehat{\mu} \in \{0, 1, \perp\}$.

We also require an FHE scheme to satisfy compactness, correctness, and security properties as follows:

Definition 2.6 (FHE Compactness). *We say that a* FHE *scheme is compact if there exists a polynomial* $\mathsf{poly}(\cdot)$ *such that for all security parameter* λ*, circuit* $C : \{0,1\}^\ell \to \{0,1\}$ *of depth at most* $\mathsf{poly}(\lambda)$*, and* $\mu_i \in \{0,1\}$ *for* $i \in [\ell]$*, the following holds: Given* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$*,* $\mathsf{ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, \mu_i)$ *for* $i \in [\ell]$*,* $\widehat{\mathsf{ct}} \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \cdots, \mathsf{ct}_\ell)$*, we always have* $|\widehat{\mathsf{ct}}| \leq \mathsf{poly}(\lambda)$.

Definition 2.7 (FHE Correctness). *We say that a* FHE *scheme is correct if for all security parameter* λ*, circuit* $C : \{0,1\}^\ell \to \{0,1\}$ *of depth at most* $\mathsf{poly}(\lambda)$*, and* $\mu_i \in \{0,1\}$ *for* $i \in [\ell]$*, the following holds: Given* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$*,* $\mathsf{ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, \mu_i)$ *for* $i \in [\ell]$*,* $\widehat{\mathsf{ct}} \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \cdots, \mathsf{ct}_\ell)$*,*

$$\Pr[\mathsf{FHE.Dec}(\mathsf{pk}, \mathsf{sk}, \widehat{\mathsf{ct}}) = f(\mu_1, \cdots, \mu_\ell)] = 1.$$

Definition 2.8 (FHE Security). *We say that a* FHE *scheme satisfies security if for all security parameter* λ*, the following holds: For any* PPT_λ *adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\mathsf{adv}_{\mathcal{A},\mathsf{FHE}} := 2 \cdot |1/2 - \Pr[\mathsf{Expt}_{\mathcal{A},\mathsf{FHE}}(1^\lambda) = 1]| < \mathsf{negl}(\lambda)$$

, where the experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{FHE}}$ *is defined as follows:*

$\mathsf{Expt}_{\mathcal{A},\mathsf{FHE}}(1^\lambda)$:

1. *On input the security parameter* 1^λ*, the challenger runs* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$*, and* $\mathsf{ct} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, b)$ *for* $b \xleftarrow{\$} \{0,1\}$*. It provides* $(\mathsf{pk}, \mathsf{ct})$ *to* $\mathcal{A}$.
2. $\mathcal{A}$ *outputs a guess* b'*. The experiment outputs* 1 *if and only if* $b = b'$.

Remark 2.9. The security definition above is in terms of PPT_λ adversaries and negl advantage, but it can be lifted to a $(\mathcal{T}, \epsilon)$ setting similar to Remark 2.4 as follows: We say that a FHE scheme is $(\mathcal{T}, \epsilon)$-secure if there exists an integer λ_0, such that for all $\lambda > \lambda_0$, for any non-uniform probabilistic $\mathcal{T}(\lambda)$ time adversary, the advantage $\mathsf{adv}_{\mathcal{A},\mathsf{FHE}}$ of the above experiment is at most $\epsilon(\lambda)$.

Additional properties. Our construction of robust streaming algorithms will be based on an FHE scheme satisfying a few additional properties. We refer to this type of FHE scheme as a *pseudorandom* FHE *(*PFHE*) scheme.*

Definition 2.10 (Pseudorandom FHE (PFHE)). *Let* FHE *be a fully homomorphic encryption scheme* (Definition 2.5)*. We call* FHE *a* $(\mathcal{T}, \epsilon)$*-pseudorandom* FHE *scheme if for all security parameter* λ*, it satisfies the following additional properties:*

- $\mathsf{FHE.Eval}$ *is deterministic.*
- (Pseudorandom pk)*. There exists an explicit distribution* $\mathcal{D}_{\widetilde{\mathsf{pk}}(\lambda)}$*, such that the following two distributions are* $(\mathcal{T}, \epsilon)$*-indistinguishable:*

$$\{\mathsf{pk} : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)\} \approx_{(\mathcal{T},\epsilon)} \mathcal{D}_{\widetilde{\mathsf{pk}}(\lambda)}$$

- (Pseudorandom ct)*. There exists an explicitly distribution* $\mathcal{D}_{\widetilde{\mathsf{ct}}(\lambda)}$*, such that the following two distributions are* $(\mathcal{T}, \epsilon)$*-indistinguishable:*

$$\left\{ \mathsf{ct} : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda) \\ \mu \in \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, \mu) \end{array} \right\} \approx_{(\mathcal{T},\epsilon)} \mathcal{D}_{\widetilde{\mathsf{ct}}(\lambda)}$$

- ($\mathbb{Z}$ Linear Homomorphism)*. Given* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$*, the set of all ciphertexts is a subset of a vector space over* $\mathbb{Z}_q$ *for some* $q = q(\lambda)$*. Let* $\mathcal{M} = \sum_{i\in[k]} x_i \cdot \mathsf{ct}_i \mod q$ *be an arbitrary linear combination of ciphertexts. Denote* $\mu_i \leftarrow \mathsf{FHE.Dec}(\mathsf{pk}, \mathsf{sk}, \mathsf{ct}_i)$*. Suppose* $\mathcal{M}$ *satisfies:*
  1. $x_i \in \mathbb{Z}$ *for all* $i \in [k]$
  2. $k \cdot \max_{i\in[k]} x_i \ll q$

  then we have the followings: $|\mathcal{M}| \le \mathsf{poly}(\lambda)$*. And there exists a* $\mathsf{poly}(\lambda)$*-time decoding algorithm* $\mathsf{LinearDec}$*, such that* $\Pr[\mathsf{LinearDec}(\mathsf{pk}, \mathsf{sk}, \mathcal{M}) = \sum_{i\in[k]} x_i \cdot \mu_i] = 1$.

As in a standard FHE *scheme, we also require that a* PFHE *scheme satisfies compactness* (Definition 2.6)*, correctness* (Definition 2.7)*, and security* (Definition 2.8)*.*

There exist FHE constructions based on LWE and Ring-LWE [33,12,13] assumptions. We will focus on the GSW scheme in our streaming construction and the BV scheme in the distributed setting. Both of them satisfy the above definition (or a slight variant) of pseudorandom FHE.

2.3 Streaming Algorithms

We review the streaming model, the structure of a streaming algorithm, and the notion of white-box adversarial robustness.

Notation. We denote the input dimension parameter of an algorithm by n. We write PPT_λ to denote probabilistic poly-time in λ.

Streaming Model. The streaming model captures key resource requirements of algorithms for database, machine learning, and network tasks, where the size of the data is significantly larger than the available storage. This model was formalized by Alon, Matias, and Szegedy [5] as a vector undergoing additive *updates* to its coordinates. In this model, the input data takes the form of a sequence of updates. These updates assume the existence of a fixed-length underlying vector which represents the current dataset, and dynamically modify this vector one coordinate at a time.

Given a dimension parameter $n \in \mathbb{N}$, the model behaves as follows:

- At the beginning of the stream, an underlying vector $\boldsymbol{x}$ is initialized to 0^n.
- The input stream is a sequence of $T \leq \mathsf{poly}(n)$ additive updates $(x_1, \cdots, x_T)$. Each update x_t for $t \in [T]$ is a tuple $x_t := (i_t, \Delta_t)$, where $i_t \in [n]$ denotes a coordinate of $\boldsymbol{x}$ and Δ_t is an arbitrary value.
  $x_t := (i_t, \Delta_t)$ is interpreted as updating $\boldsymbol{x}_{i_t} \leftarrow \boldsymbol{x}_{i_t} + \Delta_t$. Thus at any time $t' \in [T]$ during the stream, $\boldsymbol{x}$ is the accumulation of all past updates, with its i-th coordinate for $i \in [n]$: $\boldsymbol{x}_i = \sum_{t \leq t'} \mathbb{1}(i_t = i_{t_0}) \cdot \Delta_t$.

A streaming algorithm receives the stream of updates and uses limited memory to compute a function of $\boldsymbol{x}$. We stress that during the stream, the underlying vector $\boldsymbol{x}$ is not explicitly available to the algorithm, but is instead implicitly defined by the sequence of past updates.

In this work, we consider a bounded integer stream, which assumes that throughout the stream, $\boldsymbol{x}$ is promised to be in the range $\{-N, -N+1, \cdots, N-1, N\}^n$ for some integer $N \in \mathsf{poly}(n)$.

Streaming Algorithm. A streaming algorithm maintains some internal state during the stream, which evolves upon receiving new updates. It takes one pass over all updates and then uses its maintained state to compute a function of the underlying dataset $\boldsymbol{x}$. In the streaming model, the dimension parameter n (the length of $\boldsymbol{x}$) is typically large. Therefore, a streaming algorithm usually optimizes the size of the state such that it takes sublinear bits of space in n.

We define the general framework for streaming algorithms as follows:

Definition 2.11 (Streaming Algorithm). *Given a query function ensemble $\mathcal{F} = \{f_n : \mathbb{Z}^n \to X\}_{n \in \mathbb{N}}$ for some domain X, a streaming algorithm* $\mathsf{StreamAlg}$ *that computes $\mathcal{F}$ contains a tuple of operations* $\mathsf{StreamAlg} = (\mathsf{StreamAlg.Setup}, \mathsf{StreamAlg.Update}, \mathsf{StreamAlg.Report})$*, with the following properties:*

- $\mathsf{StreamAlg.Setup}(n)$ *: On input a dimension parameter n, the setup operation initializes an internal data structure $\mathcal{DS}$ within the memory of* $\mathsf{StreamAlg}$*.*
- $\mathsf{StreamAlg.Update}(x_t)$ *: On input an additive update $x_t := (i_t, \Delta_t)$, the update operation modifies $\mathcal{DS}$.*

– $\mathsf{StreamAlg.Report}() \to r$: *The report operation uses $\mathcal{DS}$ to compute a query response $r \in X$.*
A response r is said to be correct if $r = f_n(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{Z}^n$ is the accumulation of all past updates such that $\boldsymbol{x}_i = \sum_t \mathbb{1}(i_t = i) \cdot \Delta_t$ for $i \in [n]$.

In applications, StreamAlg.Setup is usually called once before the beginning of the stream, and StreamAlg.Update is called once on each update x_t during the stream. Depending on the use case, StreamAlg.Report may be called once or multiple times to get query responses.

In this work, we only consider exact computations of query functions ($r = f_n(\boldsymbol{x})$), though the above definition can also be generalized to approximation problems ($r \approx f_n(\boldsymbol{x})$).

White-box Adversarial Stream. In the conventional *oblivious* stream model, the sequence of updates maybe chosen adversarially, but is assumed to be chosen independently of the randomness of a streaming algorithm StreamAlg. In contrast, in the white-box adversarial streaming model, a sequence of stream updates $(x_1, \cdots, x_T)$ is adaptively generated by an adversary who sees the *full internal state* of StreamAlg at all times, including $\mathcal{DS}$ and any other parameters and randomness used by StreamAlg. We say that a streaming algorithm is *white-box adversarially robust* if it guarantees to output correct responses even against such powerful adversaries. This is described as the following game:

Definition 2.12 (White-Box Adversarially Robustness). *We say a streaming algorithm* StreamAlg *is white-box adversarially robust (WAR) if for all dimension parameters $n \in \mathbb{N}$, the following holds: For any PPT adversary $\mathcal{A}$, the following experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{StreamAlg}}(n)$ outputs 1 with probability at most $\mathsf{negl}(n)$:*

$\mathsf{Expt}_{\mathcal{A},\mathsf{StreamAlg}}(n)$:

1. *The challenger and $\mathcal{A}$ agree on a query function ensemble $\mathcal{F} = \{f_n : \mathbb{Z}^n \to X\}_{n\in\mathbb{N}}$ for some domain X. On input a dimension parameter n, let $f_n \in \mathcal{F}$ be the query function of the experiment.*
2. *The challenger runs* $\mathsf{StreamAlg.Setup}(n)$*. It then provides all internal states of* StreamAlg *to $\mathcal{A}$, including $\mathcal{DS}$ and any other parameters and randomness previously used by* StreamAlg.
3. *For some $T \in \mathsf{poly}(n)$, at each time $t \in [T]$:*
   - *$\mathcal{A}$ outputs an adaptive update $x_t := (i_t, \Delta_t)$ for $t \in [T]$. Also, $\mathcal{A}$ may issue a query.*
   - *The challenger runs* $\mathsf{StreamAlg.Update}(x_t)$*, and outputs a response $r_t \leftarrow$* $\mathsf{StreamAlg.Report}()$ *if it is queried.*
     The challenger then provides all internal states of StreamAlg *to $\mathcal{A}$, including $\mathcal{DS}$ and any other parameters and randomness used by* StreamAlg.
4. *The experiment outputs 1 if and only if at some time $t \in [T]$ in the stream,* StreamAlg *output an incorrect response $r_t \neq f_n(\boldsymbol{x})$, where $\boldsymbol{x}$ is the underlying data vector with $\boldsymbol{x}_i = \sum_{j\in[t]} \mathbb{1}(i_j = i) \cdot \Delta_j$ for $i \in [n]$.*

3 Streaming K-Sparse Recovery

Sparse Recovery Problem We study the sparse recovery problem for vectors in white-box adversarial streaming settings. There exist deterministic (thus WAR) streaming algorithms that recover k-sparse vectors *assuming that the input vector is k-sparse* . However, their outputs can be arbitrary when the input violates such an assumption. In contrast, our algorithm has the crucial property that it can detect if the input violates the sparsity constraint. We describe the problem as follows:

Definition 3.1 (k-Sparse Recovery). *Given a sparsity parameter k, the query function ensemble of the* k-Sparse Recovery *problem is:*

$$\mathcal{F} = \{f_n : \mathbb{Z}^n \to (\mathbb{Z}^n \cup \{\perp\})\}_{n\in\mathbb{N}} \text{ , where } f_n(\boldsymbol{x}) = \begin{cases} \boldsymbol{x} & \text{if } \|\boldsymbol{x}\|_0 \leq k \\ \perp & \text{otherwise} \end{cases}$$

Here and throughout, for the sparsity to make sense we always assume $k \leq n$. And k can be a function in n.

Before tackling this problem, we first consider a relaxation for it, which only guarantees correct recovery when the input vector $\boldsymbol{x}$ is k-sparse. In other words, we have no constraint on the function output when $\|\boldsymbol{x}\|_0 > k$.

Definition 3.2 (Relaxed k-Sparse Recovery). *Given a sparsity parameter k, $\mathcal{F} = \{f_n : \mathbb{Z}^n \to \mathbb{Z}^n\}_{n\in\mathbb{N}}$ is a query function ensemble of the* Relaxed k-Sparse Recovery *problem if for all $f_n \in \mathcal{F}$, the following holds: For all $\boldsymbol{x} \in \mathbb{Z}^n$ satisfying $\|\boldsymbol{x}\|_0 \leq k$, $f_n(\boldsymbol{x}) = \boldsymbol{x}$.*

As mentioned earlier, there exists space- and time-efficient, deterministic streaming algorithms that solve the relaxed k-sparse recovery problem. We restate the existing result as follows, which we will use as a black box subroutine in our algorithm.

Theorem 3.3 ([44,43,34]). *There exists a streaming algorithm* $\mathsf{StreamAlg}_0$ *that solves the Relaxed k-Sparse Recovery problem* (Definition 3.2)*, with* $\mathsf{StreamAlg}_0.\mathsf{Setup}, \mathsf{Update}$, *and* Report *all deterministic and run in* $\mathsf{poly}(n)$ *time.*

We will instantiate $\mathsf{StreamAlg}_0$ as concrete existing schemes and specify their space and time complexity when discussing the efficiency of our construction in Section 4. But for the purpose of proving that our construction is WAR, we only need the polynomial runtime and the deterministic property.

With the result in Theorem 4.1 and a PFHE, we are now ready to construct a WAR streaming algorithm for the (strong) k-sparse recovery problem.

Construction 3.1. *Let* $\mathsf{StreamAlg}_0$ *be an algorithm for the Relaxed k-Sparse Recovery problem from Theorem 4.1. And let* PFHE *be a pseudorandom* FHE *satisfying Definition 2.10.*

Our construction is as follows:

– $\mathsf{StreamAlg.Setup}(n)$*: Run* $\mathsf{StreamAlg}_0.\mathsf{Setup}(n)$.
Define $\ell = \lceil \log n \rceil$*. For each* $i \in [n]$*, define a circuit* $C_i : \{0,1\}^\ell \to \{0,1\}$*, such that:*

$$C_i(\mu_1, \mu_2, \cdots, \mu_\ell) = \begin{cases} 1 & \text{if } (\mu_1, \mu_2, \cdots, \mu_\ell) \text{ is the bit representation of } i \\ 0 & \text{otherwise.} \end{cases}$$

Let $\lambda = \mathcal{S}(n)$ *be the security parameter of* PFHE*, for some function* $\mathcal{S}$ *in* n *that we will specify later in Section 4. Choose a* $q \in \mathsf{poly}(n)$ *with* $q \gg n \cdot N$ *to be the modulus of the space of* PFHE *ciphertexts, where* $N \in \mathsf{poly}(n)$ *is the bound of the stream. All computations will be modulo* q*.*
Take $\mathcal{D}_{\widetilde{\mathsf{pk}}(\lambda)}$ *to be the distribution of pseudo-public key in Definition 2.10. Sample* $\widetilde{\mathsf{pk}} \leftarrow \mathcal{D}_{\widetilde{\mathsf{pk}}(\lambda)}$*.*
Similarly, take $\mathcal{D}_{\widetilde{\mathsf{ct}}(\lambda)}$ *to be the distribution of pseudo-ciphertext in Definition 2.10. Sample* $\widetilde{\mathsf{ct}_1}, \widetilde{\mathsf{ct}_2}, \cdots, \widetilde{\mathsf{ct}_\ell} \leftarrow \mathcal{D}_{\widetilde{\mathsf{ct}}(\lambda)}$*. Store* $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_\ell})$ *and sketch* $:= 0$ *in the internal data structure* $\mathcal{DS}$ *of* $\mathsf{StreamAlg}$*.*

– $\mathsf{StreamAlg.Update}(x_t)$*: Run* $\mathsf{StreamAlg}_0.\mathsf{Update}(x_t)$.
For $x_t := (i_t, \Delta_t)$*, evaluate* $\widehat{\mathsf{ct}_{i_t}} \leftarrow \mathsf{PFHE.Eval}(\widetilde{\mathsf{pk}}, C_{i_t}, \widetilde{\mathsf{ct}_1}, \widetilde{\mathsf{ct}_2}, \cdots, \widetilde{\mathsf{ct}_\ell})$*. Then update sketch as*

$$sketch \leftarrow sketch + \Delta_t \cdot \widehat{\mathsf{ct}_{i_t}}.$$

– $\mathsf{StreamAlg.Report}()$*: Let* $\boldsymbol{x}' \leftarrow \mathsf{StreamAlg}_0.\mathsf{Report}()$*. If* $\|\boldsymbol{x}'\|_0 > k$*, output* $\perp$*. Otherwise, evaluate* $\widehat{\mathsf{ct}_i} \leftarrow \mathsf{PFHE.Eval}(\widetilde{\mathsf{pk}}, C_i, \widetilde{\mathsf{ct}_1}, \widetilde{\mathsf{ct}_2}, \cdots, \widetilde{\mathsf{ct}_\ell})$ *for* $i \in [n]$*. Then compute*

$$hash \leftarrow \sum_{i \in [n]} \widehat{\mathsf{ct}_i} \cdot \boldsymbol{x}'_i.$$

Output $\boldsymbol{x}'$ *if* $hash = sketch$*, and output* $\perp$ *otherwise.*

Robustness. Recall that in the FHE Preliminary section, we define the $(\mathcal{T}, \epsilon)$-security of a FHE scheme in Remark 2.9, where $\mathcal{T}$ and ϵ are functions in λ. Also, in Construction 3.1, we set $\lambda = \mathcal{S}(n)$ for some function $\mathcal{S}$.

In this section, we will show that given appropriate relationships between these functions, the $\mathsf{StreamAlg}$ scheme in Construction 3.1 is WAR. We state our main theorem as follows:

Theorem 3.4 (WAR of Construction 3.1). *Suppose* PFHE *is a* $(\mathcal{T}, \epsilon)$ *- pseudorandom* FHE *scheme from* Definition 2.10 *with* $(\mathcal{T}, \epsilon)$*-security, and* $\mathsf{StreamAlg}_0$ *is a streaming algorithm for Relaxed* k*-Sparse Recovery from* Theorem 4.1*. If for all* $n \in \mathbb{N}$*, functions* $\mathcal{T}, \epsilon$*, and* $\mathcal{S}$ *satisfy:*

- $\mathcal{T}(\mathcal{S}(n)) = \Omega(\mathsf{poly}\, n)$
- $\epsilon(\mathcal{S}(n)) \leq \mathsf{negl}(n)$ *for some negligible functions* negl*.*

then the $\mathsf{StreamAlg}$ *scheme in* Construction 3.1 *is a WAR streaming algorithm that solves the* k*-Sparse Recovery problem* (Definition 3.1).

Roughly speaking, we need $\mathsf{poly}(n) \subseteq \mathcal{T}(\mathcal{S}(n))$ to convert between the runtime of a PFHE adversary and a StreamAlg adversary. And we need $\epsilon(\mathcal{S}(n)) \leq \mathsf{negl}(n)$ to convert between the advantage of a PFHE adversary and a StreamAlg adversary.

Proof. We first show that at any time during the stream, if the underlying vector $\boldsymbol{x}$ is k-sparse, then the query response of StreamAlg is correct.

Lemma 3.5. *Suppose* PFHE *satisfies* Definition 2.10 *and* $\mathsf{StreamAlg}_0$ *solves the Relaxed k-Sparse Recovery problem, the following holds: At any time $t \in [T]$, if the current underlying vector $\boldsymbol{x}$ satisfies $\|\boldsymbol{x}\|_0 \leq k$, then* StreamAlg.Report *correctly outputs $\boldsymbol{x}$.*

Proof. By the correctness of the $\mathsf{StreamAlg}_0$ scheme, when $\|\boldsymbol{x}\|_0 \leq k$, $\mathsf{StreamAlg}_0$.Report outputs $\boldsymbol{x}' = \boldsymbol{x}$. Since Definition 2.10 restricts PFHE.Eval to be deterministic, we essentially have $hash = \sum_{i\in[n]} \widehat{ct}_i \cdot \boldsymbol{x}'_i$ and $sketch = \sum_{i\in[n]} \widehat{ct}_i \cdot \boldsymbol{x}_i$ for the same set of ciphertexts $\widehat{ct}_i$. Thus $hash = sketch$ follows from $\boldsymbol{x}' = \boldsymbol{x}$. In this case, StreamAlg.Report correctly outputs $\boldsymbol{x}$ as specified by the query function f_n of the k-Sparsed Recovery problem.

□

It remains to show the correctness of responses in the case when the underlying data vector is denser than k. Our proof proceeds via a sequence of hybrid experiments between an adversary $\mathcal{A}$ and a challenger.

- $\mathbf{Hybrid}_0$: This is a real WAR experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{StreamAlg}}(n)$ from Definition 2.12, where the challenger runs the StreamAlg scheme from Construction 3.1. On input a dimension parameter n, the challenger samples $\widetilde{\mathsf{pk}} \leftarrow \mathcal{D}_{\widetilde{\mathsf{pk}}(\lambda)}$ and $\widetilde{\mathsf{ct}_1}, \widetilde{\mathsf{ct}_2}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}} \leftarrow \mathcal{D}_{\widetilde{\mathsf{ct}}(\lambda)}$ from the distributions of pseudo-public key and pseudo-ciphertext (Definition 2.10) and provides them to $\mathcal{A}$.
  $\mathcal{A}$ outputs $T \in \mathsf{poly}(n)$ -many adaptive updates $x_t := (i_t, \Delta_t)$ upon seeing past internal states of StreamAlg. At any time $t \in [T]$, these updates implicitly define an underlying vector $\boldsymbol{x}$, whose i-th coordinate $\boldsymbol{x}_i = \sum_t \mathbb{1}(i_t = i) \cdot \Delta_t$ for $i \in [n]$.
- $\mathbf{Hybrid}_1$: This is the same experiment as $\mathbf{Hybrid}_0$, except that $\mathcal{A}$ now runs some additional extraction on itself:
  - At each time $t \in [T]$, $\mathcal{A}$ computes the underlying vector $\boldsymbol{x}$ defined by all of its past updates.
  - $\mathcal{A}$ runs an additional instance of the $\mathsf{StreamAlg}_0$ scheme from Definition 3.2 (which solves the Relaxed k-Sparse Recovery problem):
    1. Before $\mathcal{A}$ generates its first update, it initializes its $\mathsf{StreamAlg}_0$ instance to be in the same state as the $\mathsf{StreamAlg}_0$ scheme run by the challenger. (i.e., $\mathcal{A}$ runs $\mathsf{StreamAlg}_0.\mathsf{Setup}(n)$ using the same set of randomness used by the challenger.)
    2. At each time $t \in [T]$, after $\mathcal{A}$ generates an update x_t, it calls $\mathsf{StreamAlg}_0.\mathsf{Update}(x_t)$ and then $\mathsf{StreamAlg}_0.\mathsf{Report}()$ to get a query response $\boldsymbol{x}'$.

We note that if $\mathcal{A}$ runs in $\mathsf{poly}(n)$ time in $\mathbf{Hybrid}_0$, then it still runs in $\mathsf{poly}(n)$ time in $\mathbf{Hybrid}_1$ since computing $\boldsymbol{x}$ is trivial and $\mathsf{StreamAlg}_0$ runs in poly-time as specified in Theorem 4.1.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the output of $\mathbf{Hybrid}_i$.

Lemma 3.6. *For all adversaries $\mathcal{A}$, $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = 0$.*

Proof. This just follows from the fact that the additional extractions run by $\mathcal{A}$ do not affect the output of the experiments. □

– $\mathbf{Hybrid}_2$: Same as $\mathbf{Hybrid}_1$, except that at the beginning of the experiment, the challenger samples a random coordinate $m \xleftarrow{\$} [n]$ which will be hidden from $\mathcal{A}$. The experiment outputs 1 if at some time $t \in [T]$ both of the following holds:
  1. $\mathsf{StreamAlg}$ outputs an incorrect response $r_t \neq f_n(\boldsymbol{x})$ (i.e. same condition for $\mathsf{Hyb}_1(\mathcal{A}) = 1$),
  2. The m-th coordinates of $\boldsymbol{x}$ and $\boldsymbol{x'}$ extracted by $\mathcal{A}$ satisfy $\boldsymbol{x}_m \neq \boldsymbol{x'}_m$,

We show that with the additional success condition, the success probability of $\mathbf{Hybrid}_2$ drops by at most a $1/n$ multiplicative factor compared to $\mathbf{Hybrid}_1$.

Lemma 3.7. *For all adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]/n$.*

Before proving Lemma 3.7, we make the following observation:

Claim. At any time $t \in [T]$ during the $\mathbf{Hybrid}_2$ experiment, if $\mathsf{StreamAlg}$ outputs an incorrect response $r_t \neq f_n(\boldsymbol{x})$, and $\boldsymbol{x}$ and $\boldsymbol{x'}$ are the vectors extracted by $\mathcal{A}$ at time t, then $\boldsymbol{x'} \neq \boldsymbol{x}$.

Proof. By Lemma 3.5, $r_t \neq f_n(\boldsymbol{x})$ can occur only when $\|\boldsymbol{x}\|_0 > k$ and $r_t \neq \perp$. In this case, by the construction of $\mathsf{StreamAlg.Report}$, r_t should be a k-sparse vector thus different from $\boldsymbol{x}$. Also, using deterministic $\mathsf{StreamAlg}_0.\mathsf{Update}$ and $\mathsf{StreamAlg}_0.\mathsf{Report}$, $\mathcal{A}$ exactly extracts $\boldsymbol{x'} = r_t$, so $\boldsymbol{x'} \neq \boldsymbol{x}$. □

Lemma 3.7 then follows from Claim 3:

Proof (of Lemma 3.7).. Claim 3 states that $\boldsymbol{x'} \neq \boldsymbol{x}$, and in particular, $\boldsymbol{x'}$ and $\boldsymbol{x}$ disagree at at least one coordinate. Recall that the random coordinate m is hidden from $\mathcal{A}$ at all times, so with probability at least $1/n$, m collides with a coordinate where $\boldsymbol{x'}$ and $\boldsymbol{x}$ disagrees. In this case, both conditions in $\mathbf{Hybrid}_2$ are satisfied, so $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1 \mid \mathsf{Hyb}_1(\mathcal{A}) = 1] \geq 1/n$. □

– $\mathbf{Hybrid}_3$: Same as $\mathbf{Hybrid}_2$, except that the challenger now generates an actual public key $\mathsf{pk} : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PFHE.Setup}(1^\lambda)$ to replace $\widetilde{\mathsf{pk}}$.
– $\mathbf{Hybrid}_4$: Same as $\mathbf{Hybrid}_3$, except that the challenger now uses the random coordinate $m \xleftarrow{\$} [n]$ (whose bits are denoted as $m_1 m_2 \cdots m_{\lceil \log n \rceil}$) that was sampled privately as the message in the ciphertexts: It generates $\lceil \log n \rceil$ actual ciphertexts $\mathsf{ct}_i \leftarrow \mathsf{PFHE.Enc}(\mathsf{pk}, m_i)$ for $i \in [\lceil \log n \rceil]$ to replace $(\widetilde{\mathsf{ct}_1}, \widetilde{\mathsf{ct}_2}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}})$. i.e., $(\mathsf{ct}_1, \mathsf{ct}_2, \cdots, \mathsf{ct}_{\lceil \log n \rceil})$ encrypts the bit representation of m.

Lemma 3.8. *Given that* PFHE *satisfies* Definition 2.10*, for any PPT_n adversaries $\mathcal{A}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \leq \mathsf{negl}(n)$ for some negiligble function* negl*.*

Proof. The only difference in $\mathcal{A}$'s views in $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$ is the way the challenger generates $\widetilde{\mathsf{pk}}$ or pk. In $\mathbf{Hybrid}_2$, the challenger samples $\widetilde{\mathsf{pk}} \leftarrow \mathcal{D}_{\widetilde{\mathsf{pk}}(\lambda)}$; and in $\mathbf{Hybrid}_3$, the challenger runs $\mathsf{pk} : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PFHE.Setup}(1^\lambda)$. Thus $\mathcal{A}$ distinguishing these two hybrids will distinguish the two distributions $\{\mathsf{pk} : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)\}$ and $\mathcal{D}_{\widetilde{\mathsf{pk}}(\lambda)}$ with the same advantage. And by the $(\mathcal{T}, \epsilon)$-indistinguishability in Definition 2.10, for any $\mathsf{poly}(n) \leq \mathcal{T}(\lambda)$-sized adversaries, the latter two distributions can be distinguished with probability at most $\epsilon(\lambda) \leq \mathsf{negl}(n)$. □

Lemma 3.9. *Given that* PFHE *satisfies* Definition 2.10*, for any PPT_n adversaries $\mathcal{A}$, $|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]| \leq \mathsf{negl}(n)$ for some negiligble function* negl*.*

Proof. Similar to Lemma 3.8, the only difference in $\mathcal{A}$'s views in $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_4$ is the way the challenger generates $(\widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}})$ or $(\mathsf{ct}_1, \cdots, \mathsf{ct}_{\lceil \log n \rceil})$. Recall that we have: $\{\mathsf{ct}\} \approx_{(\mathcal{T},\epsilon)} \mathcal{D}_{\widetilde{\mathsf{ct}}(\lambda)}$ from Definition 2.10. Therefore for any PPT_n adversary $\mathcal{A}$, the advantage of distinguishing $(\widetilde{\mathsf{ct}_1}, \widetilde{\mathsf{ct}_2}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}})$ in $\mathbf{Hybrid}_3$ from a ciphertext sequence $(\mathsf{ct}_1, \mathsf{ct}_2, \cdots, \mathsf{ct}_{\lceil \log n \rceil})$ where each $\mathsf{ct}_i \leftarrow \mathsf{PFHE.Enc}(\mathsf{pk}, m_i)$ is at most $\lceil \log n \rceil \cdot \epsilon(\lambda) = \mathsf{negl}(n)$. Thus $\mathcal{A}$ cannot distinguish between $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_4$ with non-negligible advantage, as otherwise either the pseudorandom-ct property in Definition 2.10 would be broken.

□

Now that we have related the success probability between the above consecutive hybrid experiments, we will show next that $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = 0$.

Lemma 3.10. *Given that* PFHE *satisfies* Definition 2.10*, for any PPT_n adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = 0$.*

Proof. As in the proof for Claim 3, if $\mathsf{Hyb}_4(\mathcal{A}) = 1$, at some time t we must have $r_t = \boldsymbol{x'} \neq \boldsymbol{x}$. Additionally, we have:

- $\boldsymbol{x'}_m \neq \boldsymbol{x}_m$, which is among the condition of $\mathsf{Hyb}_4(\mathcal{A}) = 1$, and
- $hash = sketch$, which is verified in StreamAlg.Report.

Recall the StreamAlg scheme computes $hash := \sum_{i\in[n]} \widehat{\mathsf{ct}_i} \cdot \boldsymbol{x'}_i$ and $sketch := \sum_{i\in[n]} \widehat{\mathsf{ct}_i} \cdot \boldsymbol{x}_i$, where $\widehat{\mathsf{ct}_i} \leftarrow \mathsf{PFHE.Eval}(\mathsf{pk}, C_i, \mathsf{ct}_1, \cdots, \mathsf{ct}_{\lceil \log n \rceil})$, for $C_i(\mu) = 1$ iff $\mu = i$. Therefore, given that $(\mathsf{ct}_1, \cdots, \mathsf{ct}_{\lceil \log n \rceil})$ encrypts m in $\mathbf{Hybrid}_4$, $\widehat{\mathsf{ct}_m}$ should encode 1, and $\widehat{\mathsf{ct}_i}$ encodes 0 for all $i \neq m$. Thus by the linear homomorphism property of PFHE, there exists a deterministic decoding algorithm such that $hash$ decodes to $\boldsymbol{x'}_m$ and $sketch$ decodes to $\boldsymbol{x}_m$. However, since $\boldsymbol{x'}_m \neq \boldsymbol{x}_m$, this essentially implies $hash \neq sketch$, contradicting the condition in StreamAlg.Report.

□

Finally, summarizing Lemma 3.6 to Lemma 3.10:

1. $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = 0$
2. $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]/n$
3. $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \mathsf{negl}(n)$
4. $|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]| = \mathsf{negl}(n)$
5. $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = 0$

We have that for any adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] \leq \mathsf{negl}(n) \cdot n = \mathsf{negl}(n)$. i.e., The real WAR game between $\mathcal{A}$ and a challenger running the $\mathsf{StreamAlg}$ scheme from Construction 3.1 outputs 1 with probability at most $\mathsf{negl}(n)$.

□

4 Efficiency

In this section, we analyze the bit complexity of the $\mathsf{StreamAlg}$ scheme in Construction 3.1, in addition to the runtime of $\mathsf{StreamAlg.Update}$ and $\mathsf{StreamAlg.Report}$ (which we refer to as *update time* and *report time*, respectively). We start by instantiating $\mathsf{StreamAlg}_0$ and PFHE used in our construction.

Notation. We use $\boldsymbol{b}_i$ to denote the standard basis vector, whose i-th entry equals 1 and 0 anywhere else. Our logarithms are in base 2.

4.1 Deterministic Relaxed Sparse Recovery

Recall that in Theorem 4.1, we state that there exists space- and time-efficient streaming algorithms for k-sparse recovery without detection (Definition 3.2). In general, given a sparse vector $\boldsymbol{x} \in \mathbb{Z}^n$, these algorithms perform some *linear measurement* $\Phi : \mathbb{Z}^n \to \mathbb{Z}^r$ on $\boldsymbol{x}$ with $r \ll n$ to get a compressive representation $\Phi(\boldsymbol{x})$, which is often referred to as a *sketch* of $\boldsymbol{x}$. This sketch is later used to reconstruct $\boldsymbol{x}$. The linearity property is useful for incrementally maintaining the sketch during the stream. For example, updating the sketch after incrementing the i-th coordinate $\boldsymbol{x}_i$ can be done as $\Phi(\boldsymbol{x}+\boldsymbol{b}_i) = \Phi(\boldsymbol{x})+\Phi(\boldsymbol{b}_i)$. When describing Φ as a matrix, we implicitly define the mapping of a vector $\boldsymbol{x}$ as the matrix-vector product $\Phi\boldsymbol{x}$.

Many existing algorithms implement the above specification using a random measurement matrix Φ. However, in our attempt to remove the random oracle assumption within sub-linear total space, we cannot afford to store an n-column random matrix during the stream. To this end, we use algorithms with deterministic construction, such that the measurements themselves incur no space overhead. We summarize two existing results and discuss their applicability for our purpose.

Theorem 4.1 ([34]). *There exists a streaming algorithm* $\mathsf{StreamAlg}_0$ *for* k*-Sparse Recovery without Detection, such that given an input parameter* n*, for an integer stream with entries bounded by* $\mathsf{poly}(n)$*:*

- $\mathsf{StreamAlg}_0.\mathsf{Setup}, \mathsf{Update}$, *and* Report *are all deterministic.*
- $\mathsf{StreamAlg}_0$ *takes* $\widetilde{\mathcal{O}}(k)$ *bits of space.*
- $\mathsf{StreamAlg}_0.\mathsf{Update}$ *runs in* $\widetilde{\mathcal{O}}(1)$.
- $\mathsf{StreamAlg}_0.\mathsf{Report}$ *runs in* $\widetilde{\mathcal{O}}(k^{1+c})$ *for an arbitrarily small constant* $c > 0$.

The algorithm in Theorem 4.1 uses a two-layer hashing as its measurement, which is capable of recovering all length-n, k-sparse vectors $\boldsymbol{x}$. It achieves nearly optimal space and time. Though such measurement is "constructed" non-explicitly using the probabilistic method. For explicit construction, we consider the following result:

Theorem 4.2 ([44]). *There exists a streaming algorithm* $\mathsf{StreamAlg}_0$ *for* k*-Sparse Recovery without Detection, such that given an input parameter* n*, for an integer stream with entries bounded by* $\mathsf{poly}(n)$*:*

- $\mathsf{StreamAlg}_0.\mathsf{Setup}, \mathsf{Update}$, *and* Report *are all deterministic.*
- $\mathsf{StreamAlg}_0$ *takes* $\widetilde{\mathcal{O}}(k)$ *bits of space.*
- $\mathsf{StreamAlg}_0.\mathsf{Update}$ *runs in* $\widetilde{\mathcal{O}}(1)$.
- $\mathsf{StreamAlg}_0.\mathsf{Report}$ *runs in* $\widetilde{\mathcal{O}}(k^2)$.

The measurement Φ in Theorem 4.2 is a $2k \times n$ Vandermonde matrix explicitly defined [1] as $\Phi_{i,j} = j^{i-1}$ over $\mathbb{Z}_p$, where $p > 2M$ is a large enough prime for $M \in \mathsf{poly}(n)$ being the bound on the magnitude of the stream entries. The reconstruction algorithm uses the same idea as the algebraic algorithm for decoding Reed-Solomon codes. It uses the sketch to construct an error-locator polynomial; the roots of this polynomial identify the signals appearing in the sparse superposition.

We note that the Vandermonde matrix is dense but structural. Thus to achieve the claimed $\widetilde{\mathcal{O}}(1)$ update time, we buffer every $2k$ stream updates and then efficiently process them in one batch. For preliminaries, we need the following result for fast multi-point evaluation of polynomials and the Transposition Principle:

Theorem 4.3 ([32]). *Let* R *be a ring, and let* $q \in R[x]$ *be a degree-*d *polynomial. Then given at most* d *distinct points* $x_1, \cdots, x_d \in R$*, all values* $q(x_1), \cdots, q(x_d)$ *can be computed using* $\mathcal{O}(d \log^2 d \log\log d)$ *total operations over* R.

Theorem 4.4 (Transposition Principle, [63]). *Let* K *be a field and* $\mathbf{A}$ *be an* $r \times w$ *matrix over* K*. Suppose there exists an arithmetic circuit of size* s *that can compute* $\mathbf{A}\boldsymbol{y}$ *for arbitrary length-*w *vector* $\boldsymbol{y}$ *over* K*. Then there exists an* $\mathcal{O}(s)$*-time algorithm that transforms this circuit to compute* $\mathbf{A}^\top \boldsymbol{x}$ *for arbitrary length-*r *vector* $\boldsymbol{x}$ *over* K*, with the size of the new circuit in* $\mathcal{O}(s)$.

Theorem 4.3 and Theorem 4.4 together show the following update time for Vandermonde measurements:

[1] For our discussion, we choose a Vandermonde matrix that is easy to describe. Though many other Vandermonde matrices also satisfy the specifications.

Proposition 4.5. *Suppose Φ is a $2k \times n$ Vandermonde matrix as described above. Let $x_1, x_2, \cdots, x_{2k}$ be a sequence of updates with $x_t := (i_t, \Delta_t)$ for each $t \in [2k]$. Define $\boldsymbol{v} = \sum_{t\in[2k]} \boldsymbol{b}_{i_t} \cdot \Delta_t$, then $\Phi(\boldsymbol{v})$ can be computed using $\widetilde{\mathcal{O}}(k)$ time.*

Proof. Without loss of generality, we assume all updated coordinates $i_1, i_2, \cdots, i_{2k}$ are distinct, as one can always compress multiple updates to the same coordinate as a single update. Then $i_1, i_2, \cdots, i_{2k}$ index $2k$ columns of Φ and $2k$ coordinates of $\boldsymbol{v}$ (which are the only coordinates that can possibly be non-zero). Let $\mathbf{A}$ be the indexed $2k \times 2k$ column sub-matrix and $\boldsymbol{v'}$ be the $2k$-length vector containing only $i_1, i_2, \cdots, i_{2k}$-th entries of $\boldsymbol{v}$ in their original order. We have $\Phi\boldsymbol{v} = \mathbf{A}\boldsymbol{v'}$.

Observe that the column submatrix $\mathbf{A}$ is still a Vandermonde matrix, and each row i of $\mathbf{A}^\top$ takes the form $[1, \alpha_i, \alpha_i^2, \cdots, \alpha_i^{2k-1}]$ for some $\alpha_i \in [n]$. For an arbitrary vector $u \in \mathbb{Z}_p^{2k}$, we interpret it as the coefficient vector of a degree-$2k$ polynomial in $\mathbb{Z}_p^{2k}$, then $\mathbf{A}^\top \boldsymbol{u}$ can be computed as evaluating the polynomial at $2k$ points $\alpha_1, \alpha_2, \cdots \alpha_{2k}$, which takes $\widetilde{\mathcal{O}}(k)$ time using the fast multi-point evaluation in Theorem 4.1. By the Transposition Principle, this implies asymptotically the same runtime for evaluating $\mathbf{A}\boldsymbol{v'}$ thus $\Phi\boldsymbol{v}$.

□

Therefore, once we buffer the updates into $2k$-sized batches and amortize the cost of processing each batch to the future $2k$ updates, we get $\widetilde{\mathcal{O}}(1)$ amortized runtime for $\mathsf{StreamAlg}_0.\mathsf{Update}$.

4.2 Pseudorandom FHE

We focus on GSW [33], a standard LWE based fully homomorphic encryption scheme, for our instantiation. In this section, we first show that GSW has the additional properties that we require for a PFHE (Definition 2.10). Then we specify our security parameter λ, assumption, and functions $(\mathcal{T}, \epsilon)$, and state the space and time complexity of PFHE given these parameters.

GSW Satisfies PFHE *Properties.* To start with, we briefly review the construction of GSW. We omit Dec since it is not involved in our scheme. Instead, we will describe a decoding algorithm LinearDec for linear combinations of ciphertexts (which largely resembles the original Dec of GSW) when showing the $\mathbb{Z}$ Linear Homomorphism property of GSW.

- $\mathsf{Setup}(1^\lambda)$: Choose a modulus q, a lattice dimension parameter g, and error distribution χ parametrized by λ. Also, choose some $h = \Theta(g \log q)$. Let $params = (g, h, q, \chi)$ be parameters for $\mathsf{LWE}_{g,h,q,\chi}$.
- $\mathsf{SKGen}(params)$: Sample $\overline{\boldsymbol{s}} \leftarrow \mathbb{Z}_q^g$. Output $\mathsf{sk} = \boldsymbol{s}^\top = [-\overline{\boldsymbol{s}}^\top \mid 1]$.
- $\mathsf{PKGen}(params, \mathsf{sk})$: Sample a matrix $\overline{\mathbf{A}} \leftarrow \mathbb{Z}_q^{g\times h}$ uniformly and an error vector $\boldsymbol{e} \leftarrow \chi^h$. Let $\boldsymbol{\alpha} = \overline{\boldsymbol{s}}^\top \cdot \overline{\mathbf{A}} + \boldsymbol{e}^\top$. Set $\mathsf{pk} = \mathbf{A}$ to be the $(g+1)$-row matrix consisting of g rows of $\overline{\mathbf{A}}$ followed by $\boldsymbol{\alpha}$.
- $\mathsf{Enc}(params, \mathsf{pk}, \mu)$: Sample a uniform matrix $\mathbf{R} \leftarrow \{0,1\}^{h\times h}$. For $\mathsf{pk} = \mathbf{A}$, output $\mathbf{A} \cdot \mathbf{R} + \mu \cdot \mathbf{G}$ where $\mathbf{G}$ is a gadget matrix.

Similar to the previous Remark 2.9, we assume that our set of LWE parameters *params* is $(\mathcal{T}, \epsilon)$-secure, such that for any $\mathcal{T}(\lambda)$ time adversary, the advantage of distinguishing an LWE sample from a uniformly random sample is at most $\epsilon(\lambda)$. We now check that GSW satisfies the additional properties required by PFHE:

- Eval is deterministic by the construction of GSW.
- (Pseudorandom pk, ct.) This is just the built-in security of GSW shown via the LWE hardness and the Left-over Hash lemma. Assuming the $(\mathcal{T}, \epsilon)$-hardness of $\mathsf{LWE}_{g,h,q,\chi}$, we have:

$$\left\{ (\mathbf{A}, \mathbf{A} \cdot \mathbf{R} + \mu \cdot \mathbf{G}) : \begin{matrix} \mathbf{A} \leftarrow \mathsf{PKGen} \\ \mathbf{R} \leftarrow \{0,1\}^{h \times h} \\ \mu \in \{0,1\} \end{matrix} \right\} \approx_{(\mathcal{T},\epsilon)} \left\{ (\mathbf{U}_1, \mathbf{U}_2) : \begin{matrix} \mathbf{U}_1 \leftarrow \mathbb{Z}_q^{(g+1)\times h} \\ \mathbf{U}_2 \leftarrow \mathbb{Z}_q^{(g+1)\times h} \end{matrix} \right\}.$$

- ($\mathbb{Z}$ Linear Homomorphism.) Consider the following arbitrary linear combination, where each ciphertext $\mathsf{ct}_i = \mathbf{A} \cdot \mathbf{R}_i + \mu_i \cdot \mathbf{G}$ for $i \in [k]$:

$$\mathcal{M} = \sum_{i \in [k]} x_i \cdot \mathsf{ct}_i = \mathbf{A} \cdot (\sum_{i \in [k]} x_i \mathbf{R}_i) + (\sum_{i \in [k]} x_i \mu_i) \cdot \mathbf{G}$$

  For $k \cdot \max_{i \in [k]} x_i \ll q$, it can be deterministically decrypted using just the normal GSW decryption for integer message. For completeness, we briefly describe $\mathsf{LinearDec}$ as follows:
  - $\mathsf{LinearDec}(\mathsf{pk}, \mathsf{sk}, \mathcal{M})$: Compute

$$\begin{aligned} \mathsf{sk} \cdot \mathcal{M} &= [-\overline{\boldsymbol{s}}^\top \mid 1] \cdot \mathbf{A} \cdot (\sum_{i \in [k]} x_i \mathbf{R}_i) + (\sum_{i \in [k]} x_i \mu_i) \cdot [-\overline{\boldsymbol{s}}^\top \mid 1] \cdot \mathbf{G} \\ &= \boldsymbol{e}^\top \cdot (\sum_{i \in [k]} x_i \mathbf{R}_i) + (\sum_{i \in [k]} x_i \mu_i) \cdot [-\overline{\boldsymbol{s}}^\top \mid 1] \cdot \mathbf{G} \\ &\approx (\sum_{i \in [k]} x_i \mu_i) \cdot [-\overline{\boldsymbol{s}}^\top \mid 1] \cdot \mathbf{G} \end{aligned}$$

    Since it is easy to solve LWE with respect to $\mathbf{G}$, we can recover $(\sum_{i \in [k]} x_i \mu_i) \cdot [-\overline{\boldsymbol{s}}^\top \mid 1]$ and hence $\sum_{i \in [k]} x_i \mu_i$ (modulo q).

Efficiency Based on Subexponential LWE We first specify the FHE parameters that we use under the subexponential LWE assumption.

Choose the lattice dimension parameter $g = \lambda$, the modulus q and the noise σ with subexponential modulus-to-noise ratio, and the sample complexity $h \in \Theta(g \cdot \log q)$. The subexponential hardness of LWE requires the below indistinguishability to hold for adversaries of size $\mathcal{T}(\lambda) = 2^{\lambda^\beta}$ for some constant $\beta > 0$, with subexponentially small distinguishing advantage $\epsilon(\lambda) = 2^{-\lambda^c}$ for some constant $c > 0$ denoted by $\approx_\epsilon$:

$$\left\{ (\mathbf{A}, \boldsymbol{s} \cdot \mathbf{A} + \boldsymbol{e}) \mod q : \begin{matrix} \mathbf{A} \leftarrow \mathbb{Z}_q^{g \times h} \\ \boldsymbol{s} \leftarrow \mathbb{Z}_q^{1 \times g} \\ \boldsymbol{e} \leftarrow \chi_\sigma^{1 \times h} \end{matrix} \right\} \approx_\epsilon \left\{ (\mathbf{A}, \boldsymbol{u}) : \begin{matrix} \mathbf{A} \leftarrow \mathbb{Z}_q^{g \times h} \\ \boldsymbol{u} \leftarrow \mathbb{Z}_q^{1 \times g} \end{matrix} \right\}.$$

Set the security parameter be $\lambda = \log^r n$ for some constant $r > 1/\beta$ and $r > 1/c$. We can verify this setup satisfies the relationship between functions required by the Robustness Theorem 3.4: $\mathcal{T}(\log^r n) = 2^{(\log n)^{r\beta}} = \Omega(\mathsf{poly}\, n)$ and $\epsilon(\log^r n) = 2^{-(\log n)^{rc}} \leq \mathsf{negl}(n)$ for some negligible functions negl.

Given these parameters, using GSW to implement PFHE, we have the following complexity:

Proposition 4.6. *Let $\mathcal{R}(\mathcal{A})$ denote the runtime of an algorithm $\mathcal{A}$. Assuming the above subexponential hardness of LWE, we have the following complexity regarding* $\mathsf{StreamAlg}$ *in Construction 3.1:*

1. $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}})$ *and sketch can be stored using $\widetilde{\mathcal{O}}(1)$ total bits of space.*
2. $\mathcal{R}(\mathsf{StreamAlg.Update}) = \widetilde{\mathcal{O}}(1) + \mathcal{R}(\mathsf{StreamAlg}_0.\mathsf{Update})$.
3. $\mathcal{R}(\mathsf{StreamAlg.Report}) = \widetilde{\mathcal{O}}(k) + \mathcal{R}(\mathsf{StreamAlg}_0.\mathsf{Report})$.

Proof. Recall that computations in $\mathsf{StreamAlg}$ are modulo q with $q \in \mathsf{poly}(n)$ and $q \gg n \cdot N$, where $N \in \mathsf{poly}(n)$ is the bound of the stream. Thus all entries have bit complexity in $\mathcal{O}(\log n)$.

1. $\widetilde{\mathsf{pk}}, \mathit{sketch}$, and $\widetilde{\mathsf{ct}_i}$ for $i \in \lceil \log n \rceil$ are all $\mathcal{O}(g) \times \mathcal{O}(h)$ matrices over $\mathbb{Z}_q$. For $g = \lambda \in \mathsf{poly}(\log n)$ and $h \in \Theta(g \cdot \log q)$, they can be stored using $\mathcal{O}(\mathsf{poly}(\log n)) = \widetilde{\mathcal{O}}(1)$ total bits of space.
2. The second bullet point considers the following operation:
   - Evaluate $\widehat{\mathsf{ct}_{i_t}} \leftarrow \mathsf{PFHE.Eval}(\widetilde{\mathsf{pk}}, C_{i_t}, \widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_\ell})$. Then update *sketch* as $\mathit{sketch} \leftarrow \mathit{sketch} + \Delta_t \cdot \widehat{\mathsf{ct}_{i_t}}$.

   Updating *sketch* takes $\widetilde{\mathcal{O}}(1)$ time. For evaluation, recall that C_{i_t} is a point function checking whether the input bits equal i_t. This can be naïvely implemented using $\mathcal{O}(\log n)$ arithmetic operations, each of which runs in $\mathsf{poly} \log n$ time as described in Eval_+ and $\mathsf{Eval}_\times$ above. Thus the total runtime is $\widetilde{\mathcal{O}}(1)$.
3. The third bullet point considers the following operation:
   - (If $\|\boldsymbol{x}'\|_0 > k$, output $\perp$. Otherwise:) Evaluate $\widehat{\mathsf{ct}_i} \leftarrow \mathsf{PFHE.Eval}(\widetilde{\mathsf{pk}}, C_i, \widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_\ell})$ for $i \in [n]$. Then compute $\mathit{hash} \leftarrow \sum_{i \in [n]} \widehat{\mathsf{ct}_i} \cdot \boldsymbol{x}'_i$.

   Observe that although the construction is described as evaluating n-many $\widehat{\mathsf{ct}_i}$, one for each coordinate of $\boldsymbol{x}'$, at this line of the algorithm $\boldsymbol{x}'$ is guaranteed to have at most k non-zero entries. Thus k evaluations (corresponding to non-zero coordinates of $\boldsymbol{x}'$) suffice for computing the linear combination *hash*. Each evaluation is $\widetilde{\mathcal{O}}(1)$, so the total runtime is $\widetilde{\mathcal{O}}(k)$.

□

Efficiency Based on Polynomial LWE Under the polynomial LWE assumption, we set $\lambda = n^w$ for an arbitrarily small constant $w > 0$. The $(\mathcal{T}, \epsilon)$-security then reduces to the standard definition in terms of PPT adversaries and negligible advantage. We again choose $g = \lambda$ and $h \in \Theta(g \cdot \log q)$. The complexity is:

Proposition 4.7. *Let $\mathcal{R}(\mathcal{A})$ denote the runtime of an algorithm $\mathcal{A}$. Assuming the polynomial hardness of LWE, we have the following complexity regarding* StreamAlg *in Construction 3.1:*

1. $(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}})$ *and sketch can be stored in* $\widetilde{\mathcal{O}}(n^{w_1})$ *bits of space.*
2. $\mathcal{R}(\mathsf{StreamAlg.Update}) = \widetilde{\mathcal{O}}(n^{w_2}) + \mathcal{R}(\mathsf{StreamAlg}_0.\mathsf{Update})$.
3. $\mathcal{R}(\mathsf{StreamAlg.Report}) = \widetilde{\mathcal{O}}(k \cdot n^{w_3}) + \mathcal{R}(\mathsf{StreamAlg}_0.\mathsf{Report})$.

where w_1, w_2, and w_3 are arbitrarily small positive constants.

Proof. The proof is exactly the same as the proof for Proposition 4.6, except that now the dimension of the ciphertext matrices blows up from $\mathsf{poly}\log n$ to n^w for an arbitrarily small $w > 0$.

□

4.3 Complexity of Construction 3.1

Putting together Section 4.1 and 4.2, we have the following total complexity:

Corollary 4.8. *Assuming the subexponential hardness of LWE, the* StreamAlg *scheme in Construction 3.1 has the following complexity:*

- StreamAlg *takes* $\widetilde{\mathcal{O}}(k)$ *bits of space.*
- StreamAlg.Update *runs in time* $\widetilde{\mathcal{O}}(1)$.
- StreamAlg.Report *runs in time* $\widetilde{\mathcal{O}}(k^{1+c})$ *for an arbitrarily small constant* $c > 0$.

Corollary 4.8 follows from the efficiency of the hash-based $\mathsf{StreamAlg}_0$ from Theorem 4.1 and GSW. We summarize all combinations of instantiations in Table 2 below.

Table 2. A summary of the time and space complexities of the StreamAlg scheme in Construction 2.1, given different instantiations of $\mathsf{StreamAlg}_0$ and PFHE. w denotes an arbitrarily small positive constant.

Rec. Scheme	LWE Assumption	Space	Update Time	Report Time
2-layer hash.	subexp	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(1)$	$\widetilde{\mathcal{O}}(k^{1+w})$
Vandermonde	subexp	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(1)$	$\widetilde{\mathcal{O}}(k^2)$
2-layer hash.	poly	$\widetilde{\mathcal{O}}(k + n^w)$	$\widetilde{\mathcal{O}}(n^w)$	$\widetilde{\mathcal{O}}(k \cdot (k^w + n^w))$
Vandermonde	poly	$\widetilde{\mathcal{O}}(k + n^w)$	$\widetilde{\mathcal{O}}(n^w)$	$\widetilde{\mathcal{O}}(k \cdot (k + n^w))$

References

1. Aharonov, D., Regev, O.: Lattice problems in np conp. J. ACM **52**, 749–765 (2005)
2. Ajtai, M., Braverman, V., Jayram, T., Silwal, S., Sun, A., Woodruff, D.P., Zhou, S.: The white-box adversarial data stream model. In: Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. pp. 15–27. PODS '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3517804.3526228, `https://doi.org/10.1145/3517804.3526228`
3. Almeida, A., Brás, S., Sargento, S., e.a.: Time series big data: a survey on data stream frameworks, analysis and algorithms. J Big Data (2023). https://doi.org/https://doi.org/10.1186/s40537-023-00760-1
4. Alon, N., Ben-Eliezer, O., Dagan, Y., Moran, S., Naor, M., Yogev, E.: Adversarial laws of large numbers and optimal regret in online classification. In: STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing. pp. 447–455 (2021)
5. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing. pp. 20–29. STOC '96, Association for Computing Machinery, New York, NY, USA (1996). https://doi.org/10.1145/237814.237823, `https://doi.org/10.1145/237814.237823`
6. Attias, I., Cohen, E., Shechner, M., Stemmer, U.: A framework for adversarial streaming via differential privacy and difference estimators. CoRR **abs/2107.14527** (2021)
7. Ben-Eliezer, O., Eden, T., Onak, K.: Adversarially robust streaming via dense-sparse trade-offs. In: 5th Symposium on Simplicity in Algorithms, SOSA (2022), (to appear)
8. Ben-Eliezer, O., Jayaram, R., Woodruff, D.P., Yogev, E.: A framework for adversarially robust streaming algorithms. SIGMOD Rec. **50**(1), 6–13 (2021)
9. Ben-Eliezer, O., Yogev, E.: The adversarial robustness of sampling. In: Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS. pp. 49–62 (2020)
10. Berinde, R., Indyk, P., Ruzic, M.: Practical near-optimal sparse recovery in the l1 norm. In: 2008 46th Annual Allerton Conference on Communication, Control, and Computing. pp. 198–205 (2008). https://doi.org/10.1109/ALLERTON.2008.4797556
11. Boyle, E., LaVigne, R., Vaikuntanathan, V.: Adversarially robust property-preserving hash functions. In: Blum, A. (ed.) 10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA. LIPIcs, vol. 124, pp. 16:1–16:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
12. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. pp. 309–325 (2012)
13. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011. pp. 505–524. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
14. Brakerski, Z., Vaikuntanathan, V.: Circuit-ABE from LWE: Unbounded attributes and semi-adaptive security. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016,

Part III. LNCS, vol. 9816, pp. 363–384. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53015-3$_1$3
15. Braverman, V., Hassidim, A., Matias, Y., Schain, M., Silwal, S., Zhou, S.: Adversarial robustness of streaming algorithms through importance sampling. In: Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems, NeurIPS (2021), (to appear)
16. Brown, P., Haas, P.J., Myllymaki, J., Pirahesh, H., Reinwald, B., Sismanis, Y.: Toward automated large-scale information integration and discovery. In: Data Management in a Connected World, Essays Dedicated to Hartmut Wedekind on the Occasion of His 70th Birthday. pp. 161–180 (2005)
17. Chakrabarti, A., Ghosh, P., Stoeckl, M.: Adversarially robust coloring for graph streams. In: 13th Innovations in Theoretical Computer Science Conference, ITCS (2022), (to appear)
18. Chan, T.M.: A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. J. ACM **57**(3), 16:1–16:15 (2010)
19. Chan, T.M., He, Q.: More dynamic data structures for geometric set cover with sublinear update time. In: 37th International Symposium on Computational Geometry, SoCG. pp. 25:1–25:14 (2021)
20. Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S.R., Reiss, F., Shah, M.A.: Telegraphcq: Continuous dataflow processing. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. p. 668. SIGMOD '03, Association for Computing Machinery, New York, NY, USA (2003). https://doi.org/10.1145/872757.872857, `https://doi.org/10.1145/872757.872857`
21. Cormode, G., Muthukrishnan, S.: What's new: finding significant differences in network data streams. IEEE/ACM Transactions on Networking **13**(6), 1219–1232 (2005). https://doi.org/10.1109/TNET.2005.860096
22. Cormode, G., Garofalakis, M.: Streaming in a connected world: Querying and tracking distributed data streams. In: Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology. p. 745. EDBT '08, Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1353343.1353442, `https://doi.org/10.1145/1353343.1353442`
23. Cormode, G., Jowhari, H.: Lp samplers and their applications: A survey. ACM Comput. Surv. **52**(1) (feb 2019). https://doi.org/10.1145/3297715, `https://doi.org/10.1145/3297715`
24. Cubuk, E.D., Zoph, B., Mané, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation policies from data. CoRR **abs/1805.09501** (2018)
25. Dasu, T., Johnson, T., Muthukrishnan, S., Shkapenyuk, V.: Mining database structure; or, how to build a data quality browser. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data. pp. 240–251 (2002)
26. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. J. Comput. Syst. Sci. **38**(1), 86–124 (1989)
27. Feng, Y., Woodruff, D.P.: Improved algorithms for white-box adversarial streams. In: ICML (2023)
28. Fiat, A., Kaplan, H.: Making data structures confluently persistent. J. Algorithms **48**(1), 16–58 (2003)
29. Flajolet, P., Nigel Martin, G.: Probabilistic counting algorithms for data base applications. Journal of Computer and System Sciences **31**(2), 182–209

(1985). https://doi.org/https://doi.org/10.1016/0022-0000(85)90041-8, https://www.sciencedirect.com/science/article/pii/0022000085900418
30. Forbes, M.A., Shpilka, A.: On identity testing of tensors, low-rank recovery and compressed sensing. In: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing. p. 163–172. STOC '12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2213977.2213995, https://doi.org/10.1145/2213977.2213995
31. Gaber, M.M.: Data Stream Processing in Sensor Networks, pp. 41–48. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). https://doi.org/10.1007/3-540-73679-4_4, https://doi.org/10.1007/3-540-73679-4_4
32. von zur Gathen, J., Gerhard, J.: Fast polynomial evaluation and interpolation. Cambridge University Press, 3 edn. (2013). https://doi.org/10.1017/CBO9781139856065.013
33. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_5
34. Gilbert, A.C., Li, Y., Porat, E., Strauss, M.J.: For-all sparse recovery in near-optimal time. ACM Trans. Algorithms **13**(3) (mar 2017). https://doi.org/10.1145/3039872, https://doi.org/10.1145/3039872
35. Goldreich, O., Goldwasser, S.: On the limits of nonapproximability of lattice problems. J. Comput. Syst. Sci. **60**(3), 540–563 (2000), https://doi.org/10.1006/jcss.1999.1686
36. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. CoRR **abs/1412.6572** (2014), http://arxiv.org/abs/1412.6572
37. Hassidim, A., Kaplan, H., Mansour, Y., Matias, Y., Stemmer, U.: Adversarially robust streaming algorithms via differential privacy. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, NeurIPS (2020)
38. Haviv, I., Regev, O.: Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 469–477. ACM Press (Jun 2007). https://doi.org/10.1145/1250790.1250859
39. Holmgren, J., Liu, M., Tyner, L., Wichs, D.: Nearly optimal property preserving hashing. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13509, pp. 473–502. Springer (2022)
40. Hu, Y., Qu, A., Wang, Y., Work, D.B.: Streaming data preprocessing via online tensor recovery for large environmental sensor networks. ACM Trans. Knowl. Discov. Data **16**(6) (jul 2022). https://doi.org/10.1145/3532189, https://doi.org/10.1145/3532189
41. Ilyas, A., Engstrom, L., Madry, A.: Prior convictions: Black-box adversarial attacks with bandits and priors. CoRR **abs/1807.07978** (2018)
42. Indyk, P.: Stable distributions, pseudorandom generators, embeddings, and data stream computation. J. ACM **53**(3), 307–323 (2006)
43. Indyk, P.: Explicit constructions for compressed sensing of sparse signals. In: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms. p. 30–33. SODA '08, Society for Industrial and Applied Mathematics, USA (2008)

44. Jafarpour, S.: Deterministic Compressed Sensing. Ph.D. thesis, Princeton University (2011)
45. Kacham, P., Pagh, R., Thorup, M., Woodruff, D.P.: Pseudorandom hashing for space-bounded computation with applications in streaming (2023)
46. Kane, D.M., Nelson, J., Woodruff, D.P.: On the exact space complexity of sketching and streaming small norms. In: Charikar, M. (ed.) Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010. pp. 1161–1178. SIAM (2010)
47. Kaplan, H.: Persistent data structures. In: Handbook of Data Structures and Applications. Chapman and Hall/CRC (2004)
48. Kaplan, H., Mansour, Y., Nissim, K., Stemmer, U.: Separating adaptive streaming from oblivious streaming using the bounded storage model. In: Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO, Proceedings, Part III. pp. 94–121 (2021)
49. Khot, S.: Hardness of approximating the shortest vector problem in lattices. In: 45th FOCS. pp. 126–135. IEEE Computer Society Press (Oct 2004). https://doi.org/10.1109/FOCS.2004.31
50. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial machine learning at scale. In: 5th International Conference on Learning Representations, ICLR, Conference Track Proceedings (2017)
51. Lall, A., Sekar, V., Ogihara, M., Xu, J., Zhang, H.: Data streaming algorithms for estimating entropy of network traffic. SIGMETRICS Perform. Eval. Rev. **34**(1), 145–156 (jun 2006). https://doi.org/10.1145/1140103.1140295, `https://doi.org/10.1145/1140103.1140295`
52. Lightstone, S.: Physical database design for relational databases. In: Encyclopedia of Database Systems, Second Edition. Springer (2018)
53. Liu, Y., Chen, X., Liu, C., Song, D.: Delving into transferable adversarial examples and black-box attacks. In: 5th International Conference on Learning Representations, ICLR, Conference Track Proceedings (2017)
54. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5$_1$
55. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations, ICLR, Conference Track Proceedings (2018)
56. Menuhin, B., Naor, M.: Keep that card in mind: Card guessing with limited memory. CoRR **abs/2107.03885** (2021)
57. Muthukrishnan, S.: Data streams: Algorithms and applications. Foundations and Trends® in Theoretical Computer Science **1**(2), 117–236 (2005). https://doi.org/10.1561/0400000002, `http://dx.doi.org/10.1561/0400000002`
58. Nisan, N.: Pseudorandom generators for space-bounded computation. Comb. **12**(4), 449–461 (1992)
59. Phillips, J.M., Verbin, E., Zhang, Q.: Lower bounds for number-in-hand multiparty communication complexity, made easy (2015)
60. Roghani, M., Saberi, A., Wajc, D.: Beating the folklore algorithm for dynamic matching. In: 13th Innovations in Theoretical Computer Science Conference, ITCS (2022), (to appear)
61. Schmidt, L., Santurkar, S., Tsipras, D., Talwar, K., Madry, A.: Adversarially robust generalization requires more data. In: Advances in Neural Information Processing

Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS. pp. 5019–5031 (2018)

62. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data. pp. 23–34. ACM (1979)
63. Shoup, V.: Fast construction of irreducible polynomials over finite fields. Journal of Symbolic Computation **17**(5), 371–391 (1994). https://doi.org/https://doi.org/10.1006/jsco.1994.1025, https://www.sciencedirect.com/science/article/pii/S074771718471025X
64. Shukla, A., Deshpande, P., Naughton, J.F., Ramasamy, K.: Storage estimation for multidimensional aggregates in the presence of hierarchies. In: VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases. pp. 522–531 (1996)
65. Smart, N., Vercauteren, F.: Fully homomorphic SIMD operations. Cryptology ePrint Archive, Report 2011/133 (2011), https://eprint.iacr.org/2011/133
66. Sung, M., Kumar, A., Li, L., Wang, J., Xu, J.: Scalable and efficient data streaming algorithms for detecting common content in internet traffic. In: 22nd International Conference on Data Engineering Workshops (ICDEW'06). pp. 27–27 (2006). https://doi.org/10.1109/ICDEW.2006.130
67. Szalay, A., Gray, J., VandenBerg, J.: Petabyte scale data mining: Dream or reality? Proceedings of SPIE - The International Society for Optical Engineering (08 2002). https://doi.org/10.1117/12.461427
68. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. International Conference on Learning Representations (2014)
69. Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I.J., Boneh, D., McDaniel, P.D.: Ensemble adversarial training: Attacks and defenses. In: 6th International Conference on Learning Representations, ICLR, Conference Track Proceedings (2018)
70. Venkataraman, S., Song, D.X., Gibbons, P.B., Blum, A.: New streaming algorithms for fast detection of superspreaders. In: Network and Distributed System Security Symposium (2005), https://api.semanticscholar.org/CorpusID:538509
71. Wajc, D.: Rounding dynamic matchings against an adaptive adversary. In: Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC. pp. 194–207 (2020)
72. Woodruff, D.P., Zhang, Q.: Tight bounds for distributed functional monitoring. In: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing. p. 941–960. STOC '12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2213977.2214063, https://doi.org/10.1145/2213977.2214063
73. Woodruff, D.P., Zhou, S.: Tight bounds for adversarially robust streams and sliding windows via difference estimators. In: 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS. pp. 1183–1196 (2021)
74. Zhao, Q.G., Kumar, A., Wang, J., Xu, J.J.: Data streaming algorithms for accurate and efficient measurement of traffic and flow matrices. SIGMETRICS Perform. Eval. Rev. **33**(1), 350–361 (jun 2005). https://doi.org/10.1145/1071690.1064258, https://doi.org/10.1145/1071690.1064258

Supplementary Material

We discuss our results in the distributed model, and our results for matrix and tensor recovery in the supplementary material.

A Distributed K-Sparse Recovery from Ring LWE

A.1 Distributed Computation

In the distributed computation model, the dataset is split over multiple servers. The goal is to design a communication protocol for servers to collectively compute a function on the aggregated dataset. In this work, we consider the message-passing coordinator model (see, e.g., [59]), where there are T servers and a single central coordinator. Each server $s_t : t \in [T]$ receives a partition of data in the form of an n-dimensional vector $\boldsymbol{x}_t$, and communicates with the central coordinator through a two-way, single-round communication channel. At the end of communication, the coordinator needs to compute or approximate a function $f_n(\boldsymbol{x})$ of the aggregated vector $\boldsymbol{x} = \sum_{t\in[T]} \boldsymbol{x}_t$. We aim to design protocols that minimize the total amount of communication through the channel and speed up the computation at each server and the coordinator. See also [72] and references therein on the distributed model.

Similar to the streaming case, we focus on designing a robust protocol in a white-box adversarial distributed setting, where the dataset is generated and partitioned to each server by a white-box adversary who monitors the internal state of the coordinator and the servers. We define the model as follows:

Definition A.1 (Distributed Protocol). *Given a query function ensemble $\mathcal{F} = \{f_n : \mathbb{Z}^n \to X\}_{n\in\mathbb{N}}$ for some domain X, a distributed protocol* $\mathsf{DistProt}$ *that computes $\mathcal{F}$ contains a tuple of algorithms* $\mathsf{DistProt} = (\mathsf{DistProt.Setup}, \mathsf{DistProt.Server}, \mathsf{DistProt.Coordinator})$*, with the following properties:*

- $\mathsf{DistProt.Setup}(n, \rho_0) \to \zeta$ *: On input a parameter n, the coordinator samples randomness ρ_0 and uses ρ_0 to compute some state ζ. It sends ζ to all servers.*
- $\mathsf{DistProt.Server}(\zeta, \boldsymbol{x}_t, \rho_t) \to p_t$ *: On input a state ζ and a data partition $\boldsymbol{x}_t \in \mathbb{Z}^n$, the t-th server samples randomness ρ_t and uses it to compute a packet p_t. It sends p_t back to the coordinator.*
- $\mathsf{DistProt.Coordinator}(p_1, \cdots, p_T, \rho_c) \to r$*: On input T packets $p_1, \cdots, p_T$, the coordinator samples randomness ρ_c and computes a query response $r \in X$. A response r is said to be correct if $r = f_n(\boldsymbol{x})$ where $\boldsymbol{x} = \sum_{t\in[T]} \boldsymbol{x}_t$.*

As in the streaming setting, we again assume bounded integer inputs, such that an arbitrary subset sum of $\{\boldsymbol{x}_t : t \in [T]\}$ is in $[-N, N]^n$ for some $N \in \mathsf{poly}(n)$.

Definition A.2. *We say that a distributed protocol* $\mathsf{DistProt}$ *is white-box adversarially robust if for all dimension parameters $n \in \mathbb{N}$, the following holds: For*

any PPT adversary $\mathcal{A}$, the following experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{DistProt}}(n)$ outputs 1 with probability at most $\mathsf{negl}(n)$:

$\mathsf{Expt}_{\mathcal{A},\mathsf{DistProt}}(n)$:

1. *The challenger and $\mathcal{A}$ agree on a query function ensemble $\mathcal{F} = \{f_n : \mathbb{Z}^n \to X\}_{n\in\mathbb{N}}$ for some domain X. On input a dimension parameter n, let $f_n \in \mathcal{F}$ be the query function of the experiment.*
2. *The challenger samples all randomness ρ_0, $\rho_t : t \in [T]$, and ρ_c for the servers and the coordinator. It then provides $\rho_0, \rho_1, \cdots, \rho_T$ and ρ_c to $\mathcal{A}$.*
3. *$\mathcal{A}$ generates T partitions of data $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T \in \mathbb{Z}^n$ and provides them to the challenger.*
4. *The challenger executes the protocol to sequentially compute the following:*
   (i) $\zeta \leftarrow \mathsf{DistProt.Setup}(n, \rho_0)$.
   (ii) $p_t \leftarrow \mathsf{DistProt.Server}(\zeta, \boldsymbol{x}_t, \rho_t)$ *for all* $t \in [T]$.
   (iii) $r \leftarrow \mathsf{DistProt.Coordinator}(p_1, \cdots, p_T, \rho_c)$.
5. *The experiment outputs 1 if and only if at some time $r \neq f_n(\boldsymbol{x})$.*

Remark A.3. We remark on the connection between the streaming model and the distributed model. There exists a well-known reduction from a streaming algorithm to a distributed protocol under the oblivious setting: Each server runs $\mathsf{StreamAlg.Update}$ on the same initial state $\zeta \leftarrow \mathsf{StreamAlg.Setup}$, using all entries of $\boldsymbol{x}_t$ (its data partition) as updates. It then sends the resulting sketch to the coordinator. As long as the sketches are linearly measured, the coordinator can run $\mathsf{StreamAlg.Report}$ on the sum of all sketches to compute a query response.

One may use the same recipe to construct a *WAR* distributed protocol from a WAR streaming algorithm. In general, the robustness is retained by such reduction if $\mathsf{StreamAlg.Update}$ and $\mathsf{StreamAlg.Report}$ are deterministic. (Otherwise, adversaries in the experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{DistProt}}$ would have additional views to all "future" randomness, while adversaries in the streaming experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{StreamAlg}}$ only monitors the past randomness already used by $\mathsf{StreamAlg}$.) Our Construction 3.1 satisfies this condition.

A.2 Distributed Protocol Construction

For completeness, we briefly describe the distributed protocol for k-Sparse Recovery as follows, which essentially follows the same route as our streaming Construction 3.1.

Construction A.1. *Similar to Construction 3.1, let PFHE be a pseudorandom FHE scheme. And let $\mathsf{StreamAlg}_0$ be a streaming algorithm for the Relaxed k-Sparse Recovery problem. We assume that $\mathsf{StreamAlg}_0$ performs a deterministic linear measurement Φ_n on arbitrary length-n input vector, as described in the previous instantiation Section 4.1.*

– $\mathsf{DistProt.Setup}(n)$ *: Let $\lambda = \mathcal{S}(n)$ be the security parameter of PFHE. Sample $\widetilde{\mathsf{pk}} \leftarrow \mathcal{D}_{\widetilde{\mathsf{pk}}(\lambda)}$ and $\widetilde{\mathsf{ct}}_1, \widetilde{\mathsf{ct}}_2, \cdots, \widetilde{\mathsf{ct}}_{\lceil \log n \rceil} \leftarrow \mathcal{D}_{\widetilde{\mathsf{ct}}(\lambda)}$. Send $\zeta := (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}}_1, \cdots, \widetilde{\mathsf{ct}}_{\log n})$ to all servers*

– $\mathsf{DistProt.Server}(\zeta, \boldsymbol{x_t})$ *: Given* $\zeta := (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}})$*, evaluate* $\widehat{\mathsf{ct}_i} \leftarrow \mathsf{PFHE.Eval}(\widetilde{\mathsf{pk}}, C_i, \widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}})$ *for* $i \in [n]$*, where* C_i *is the same point function as in Construction 3.1.*
Compute $\Phi_n(\boldsymbol{x_t})$ *and* $sketch_t \leftarrow \sum_{i\in[n]} \widehat{\mathsf{ct}_i} \cdot (\boldsymbol{x_t})_i$*, where* $(\boldsymbol{x_t})_i$ *denotes the* i*-th coordinate of* $\boldsymbol{x_t}$*. Send* $p_t := (sketch_t, \Phi_n(\boldsymbol{x_t}))$ *to the coordinator.*

– $\mathsf{DistProt.Coordinator}(p_1, \cdots, p_T)$*: Let* $\boldsymbol{x'} \leftarrow \mathsf{StreamAlg}_0.\mathsf{Report}(\sum_{t\in[T]} \Phi_n(\boldsymbol{x_t}))$*, i.e. the output of running the reconstruction algorithm of* $\mathsf{StreamAlg}_0$ *on the sum of all measurement results. If* $\|\boldsymbol{x'}\|_0 > k$*, output* $\perp$*.*
Otherwise, evaluate $\widehat{\mathsf{ct}_i} \leftarrow \mathsf{PFHE.Eval}(\widetilde{\mathsf{pk}}, C_i, \widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}})$ *for* $i \in [n]$*. Then compute* $hash \leftarrow \sum_{i\in[n]} \widehat{\mathsf{ct}_i} \cdot \boldsymbol{x'_i}$*. Output* $\boldsymbol{x'}$ *if* $hash = \sum_{t\in[T]} sketch_t$*, and output* $\perp$ *otherwise.*

A.3 Efficiency

In this section, we focus on showing that we can use BV [13], a Ring-LWE based FHE scheme, to achieve near-linear process time on DistProt.Server under *polynomial* ring-LWE assumption.

As a warm-up, we state the complexity of Constrution A.1 using GSW to implement PFHE:

Proposition A.4. *Let* $\mathcal{R}(\mathcal{A})$ *denote the runtime of an algorithm or an evaluation* $\mathcal{A}$*, we have the following complexity regarding* DistProt *in Construction A.1:*

– *Assuming the subexponential hardness of LWE:*
  1. *The total bits of communication between an arbitrary server and the coordinator is* $\widetilde{\mathcal{O}}(1) + size(\Phi_n(\boldsymbol{x}))$*.*
  2. $\mathcal{R}(\mathsf{DistProt.Coordinator}) = \widetilde{\mathcal{O}}(k) + \mathcal{R}(\mathsf{StreamAlg}_0.\mathsf{Report})$*.*
  3. $\mathcal{R}(\mathsf{DistProt.Server}) = \widetilde{\mathcal{O}}(k) + \mathcal{R}(\Phi_n(\boldsymbol{x}))$*.*

– *Assuming the polynomial hardness of LWE: :*
  1. *The total bits of communication between an arbitrary server and the coordinator is* $\widetilde{\mathcal{O}}(n^{w_1}) + size(\Phi_n(\boldsymbol{x}))$*.*
  2. $\mathcal{R}(\mathsf{DistProt.Coordinator}) = \widetilde{\mathcal{O}}(k \cdot n^{w_2}) + \mathcal{R}(\mathsf{StreamAlg}_0.\mathsf{Report})$*.*
  3. $\mathcal{R}(\mathsf{DistProt.Server}) = \widetilde{\mathcal{O}}(k \cdot n^{w_3}) + \mathcal{R}(\Phi_n(\boldsymbol{x}))$*.*

where $\boldsymbol{x} \in [-N, N]^n$ *is arbitrary, and* w_1, w_2*, and* w_3 *are arbitrarily small positive constants.*

Proof. The communication complexity and the runtime of DistProt.Coordinator are the same as the bit complexity and report time of StreamAlg in Construction 3.1 (except some negligible addition operations). The runtime of DistProt.Server is roughly $n \cdot \mathcal{R}(\mathsf{StreamAlg.Update})$, because every server has to perform n evaluations, one for each coordinate of its data partition.

□

Efficiency Based on Ring-LWE. Before discussing the efficiency advantage, we briefly review some important features of BV to show that it satisfies a variant of the PFHE definition. See [13] for details of the BV construction.

Consider a ring $R := \frac{\mathbb{Z}[x]}{x^g+1}$ where g is the degree parameter. Let R_q denote R/qR for a prime modulus q, and let R_t be the message space for some prime $t \in \mathbb{Z}_q^*$ and $t \ll q/g$:

- The public keys of BV are two-dimensional vectors in R_q^2, which are computationally indistinguishable from uniformly random samples from R_q^2 under the Ring-LWE assumption.
  The ciphertexts are also vectors over R_q that are indistinguishable from uniform random.
- BV satisfies an analogy of $\mathbb{Z}$ Linear Homomorphism, with coefficients of linear combinations $\mathcal{M} = \sum_{i\in[k]} x_i \cdot \mathsf{ct}_i$ being small norm ring elements $x_i \in R_t$ as opposed to integer scalars. As long as the linear combination of corresponding messages ($\mathcal{M}_{msg} = \sum_{i\in[k]} x_i \cdot m_i$, where $m_i \in R_t$ is the message encrypted by ct_i) still lies in R_t, running the standard BV decoding algorithm on $\mathcal{M}$ output the correct message.
- Eval and Dec are deterministic by the construction of BV.

We choose parameters for our Ring-LWE based construction as follows: Let $\lambda = n^w$ for an arbitrarily small constant $w > 0$. Let the dimension $g = \lambda$, the modulus of the message space $t > n \cdot N$, where $N \in \mathsf{poly}(n)$ is the bound of the input data. The modulus q is chosen with subexponential modulus-to-noise ratio and $q \gg g \cdot t$ The Ring-LWE assumption states that for any $h \in \mathsf{poly}(\lambda)$, it holds that

$$\{(a_i, a_i \cdot s + e_i) : e_i \leftarrow \chi\}_{i\in[h]} \approx_c \{(a_i, u_i) : u_i \leftarrow R_q\}_{i\in[h]}$$

where s is sampled from the noise distribution χ and a_i are uniform in R_q.
We have the following complexity:

Proposition A.5. *Let $\mathcal{R}(\mathcal{A})$ denote the runtime of an algorithm or an evaluation $\mathcal{A}$. Assuming the polynomial hardness of Ring-LWE, a variant of* DistProt *in Construction A.1 has the following complexity:*

1. *The total bits of communication between an arbitrary server and the coordinator is $\widetilde{\mathcal{O}}(n^{w_1})$ + size$(\Phi_n(\boldsymbol{x}))$.*
2. *$\mathcal{R}(\mathsf{DistProt.Coordinator}) = \widetilde{\mathcal{O}}(n) + \mathcal{R}(\mathsf{StreamAlg}_0.\mathsf{Report})$.*
3. *$\mathcal{R}(\mathsf{DistProt.Server}) = \widetilde{\mathcal{O}}(k \cdot n_2^w) + \mathcal{R}(\Phi_n(\boldsymbol{x}))$.*

where $\boldsymbol{x} \in [-N, N]^n$ is arbitrary, and w_1, w_2 are arbitrarily small positive constants.

Proof. In comparison to the complexity under polytime standard LWE, we save a $n^{\Theta(1)}$ factor in the runtime of DistProt.Server. This is due to speeding up the following two steps:

1. Evaluate $\widehat{\mathsf{ct}_i} \leftarrow \mathsf{PFHE.Eval}(\widetilde{\mathsf{pk}}, C_i, \widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}})$ for all $i \in [n]$

2. Compute $\sum_{i\in[n]} \widehat{\mathsf{ct}_i} \cdot \boldsymbol{x}_i$

The idea is to represent the messages as ring elements over R_q as opposed to individual scalars, and apply fast Fourier transforms to efficiently multiply them.

In a variant of Construction A.1, instead of performing n evaluations one for each coordinate $i \in [n]$, we partition the dimension n into n^{1-w} chunks. Each chunk is of length n^w, where $g = n^w$ is the degree parameter of the ring for an arbitrarily small constant $w > 0$. Such that a length-n vector $\boldsymbol{x}$ is partitioned into n^{1-w} consecutive segments, each can be represented as a (small-normed in comparison to q) ring element $x_j \in R_q$ for $j \in [n^{1-w}]$.

We also define a new set of functions C'_j as follows:

$$C'_j(\mu_1, \cdots, \mu_{\lceil \log n \rceil}) =$$

$$\begin{cases} 1 & \text{if } (\mu_1, \cdots, \mu_{\lceil \log n \rceil}) \text{ represents a value in } [jn^w, (j+1)n^w) \\ 0 & \text{otherwise.} \end{cases}$$

We evaluate $\widehat{\mathsf{ct}'_j} \leftarrow \mathsf{PFHE.Eval}(\widetilde{\mathsf{pk}}, C'_j, \widetilde{\mathsf{ct}_1}, \cdots, \widetilde{\mathsf{ct}_{\lceil \log n \rceil}})$ for all $j \in [n^{1-w}]$ to replace the above step 1. The total evaluation time is $\mathcal{O}(n^{1-w} \cdot n^w \,\mathsf{poly}\log(n^w)) = \widetilde{\mathcal{O}}(n)$, since two ring elements can be multiplied in $\mathcal{O}(n^w \,\mathsf{poly}\log(n^w))$ time using FFT. Then the linear combination in the above step 2 is replaced by $\sum_{j\in[n^{1-w}]} \widehat{\mathsf{ct}'_j} \cdot x_j$, which similarly takes $\widetilde{\mathcal{O}}(n)$ time to compute.

Lastly, we check that given two vectors $\boldsymbol{x}$ and $\boldsymbol{x'}$ differ at a coordinate m, when $\mathsf{ct}_1, \cdots, \mathsf{ct}_{\lceil \log n \rceil}$ encrypt the bits of m, we still have

$$sketch = \sum_{j\in[n^{1-w}]} \widehat{\mathsf{ct}'_j} \cdot x_j \neq \sum_{j\in[n^{1-w}]} \widehat{\mathsf{ct}'_j} \cdot x'_j = hash.$$

because they can be decrypted into distinct ring elements. This concludes that the above variant of Construction A.1 remains to be correct while achieving near-linear runtime of DistProt.Coordinator and DistProt.Server.

□

B Low-Rank Matrix and Tensor Recovery

The low-rank matrix and tensor recovery problems are defined as follows:

Definition B.1 (Rank-k Matrix Recovery). *Given a rank parameter k, let $\mathbf{X} \in \mathbb{Z}^{n\times m}$ denote an input data matrix. The query function ensemble of the* Rank-k Matrix Recovery *problem is:*

$$\mathcal{F} = \{f_n : \mathbb{Z}^{n\times m} \to (\mathbb{Z}^{n\times m} \cup \{\perp\})\}_{n\geq m\in\mathbb{N}},$$

$$\textit{where } f_n(\mathbf{X}) = \begin{cases} \mathbf{X} & \textit{if } rank(\mathbf{X}) \leq k \\ \perp & \textit{otherwise} \end{cases}$$

Let $\otimes$ denote the outer product of two vectors. We use the following definition for tensor rank:

Definition B.2 (CANDECOMP/PARAFAC (CP) rank). *For a tensor $X \in \mathbb{R}^{n^d}$, consider it to be the sum of k rank-1 tensors: $X = \sum_{i=1}^{k}(x_{i1} \otimes x_{i2} \otimes \cdots \otimes x_{id})$ where $x_{ij} \in \mathbb{R}^n$. The smallest number of rank-1 tensors that can be used to express a tensor X is then defined to be the rank of the tensor.*

Definition B.3 (Rank-k Tensor Recovery). *Given a rank parameter k, let $\mathbf{X} \in \mathbb{Z}^{n^d}$ denote an input data tensor. The query function ensemble of the* Rank-k Tensor Recovery *problem is:*

$$\mathcal{F} = \{f_n : \mathbb{Z}^{n^d} \to (\mathbb{Z}^{n^d} \cup \{\perp\})\}_{n\in\mathbb{N}, d\in\mathcal{O}(1)},$$

$$\text{where } f_n(\mathbf{X}) = \begin{cases} \mathbf{X} & \text{if } rank(\mathbf{X}) \le k \\ \perp & \text{otherwise} \end{cases}$$

The low-rank matrix and tensor recovery problems are natural generalizations of the k-sparse vector recovery problems. Using the same framework as the vector case, they can be solved by maintaining two hashes: one recovers the matrix or tensor but potentially produces garbage output, and the other uses a cryptographic tool to verify whether the output matches the input data. These are natural generalizations of the k-sparse vector recovery problems.

In [27], the recovery hash is measured by a random Gaussian matrix, and the verification hash is measured by an SIS matrix. Both measurements are assumed to be heuristically compressed by an idealized hash function. We instead replace the former with a deterministic measurement, which can be explicitly constructed and decoded in polynomial time [30]. The latter is replaced by the same FHE-based hashing as the one we used for the vector case in Construction 3.1, which operates on the vectorization of the data matrix or tensor. Similar to the vector case, our construction does not require using an idealized hash function or storing a separate large state.

In the streaming model, the complexities of our matrix and tensor recovery algorithms are as follows:

Proposition B.4. *Let $\mathcal{R}(\mathcal{A})$ denote the runtime of an algorithm or an evaluation $\mathcal{A}$. Assuming the polynomial hardness of LWE,*

- *There exists a WAR streaming algorithm* StreamAlg *for the rank-k matrix recovery problem using $\widetilde{\mathcal{O}}(nk)$ bits of space.*
- *There exists a WAR streaming algorithm* StreamAlg *for the rank-k tensor recovery problem using $\widetilde{\mathcal{O}}(nk^{\lceil \log d \rceil})$ bits of space.*

Proof. The recovery scheme for matrices and tensors takes $\widetilde{\mathcal{O}}(nk)$ and $\widetilde{\mathcal{O}}(nk^{\lceil \log n \rceil})$ bits of space, respectively [30]. Under the polynomial hardness assumption of LWE, we set $\lambda = n^c$ for an arbitrarily small positive constant c. Taking the sum of the two terms produces the claimed complexities. □

In addition, the above streaming algorithms can be converted into WAR distributed protocols for matrix and vector recovery, following the same roadmap as in Section A.

Proposition B.5. *Let $\mathcal{R}(\mathcal{A})$ denote the runtime of an algorithm or an evaluation $\mathcal{A}$. Assuming the polynomial hardness of LWE: :*

- *There exists a WAR distributed protocol* DistProt *for the rank-k matrix recovery problem, using $\widetilde{\mathcal{O}}(nk)$ bits of communication between each server and the coordinator, and $\widetilde{\mathcal{O}}(n^2k)$ processing time at each server.*
- *There exists a WAR streaming algorithm* StreamAlg *for the rank-k tensor recovery problem, using $\widetilde{\mathcal{O}}(nk^{\lceil \log d \rceil})$ bits of communication between each server and the coordinator.*

Proof. The total bits of communication between each server and the coordinator equals the size of the state in the above streaming algorithms. For the low-rank matrix recovery, the deterministic scheme performs $\widetilde{\mathcal{O}}(nk)$ measurements using n-sparse matrices, and the runtime for computing the FHE-based hash is comparably small. This gives $OO(n^2k)$ processing time at each server. □

We remark that the rank-k matrix recovery algorithm in the previous work [27] requires $\widetilde{\mathcal{O}}(n^3k)$ processing time in the distributed setting, which our construction improves upon.