Fast White-Box Adversarial Streaming Without a Random Oracle

Ying Feng [1] **Aayush Jain** [1] **David P. Woodruff** [1]

Abstract

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 focus on the sparse recovery problem and extend our result to other tasks such as distinct element estimation and low-rank approximation of matrices and tensors. The main drawback of 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. Also, the previous work suffers from large update time. We construct a near-optimal solution for the sparse recovery problem in white-box adversarial streams, based on the subexponentially secure Learning with Errors assumption. Importantly, our solution does not require a random oracle and has a polylogarithmic per item processing time. We also give results in a related white-box adversarially robust 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 (Alon et al., 1996), 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 (Lall et al., 2006; Zhao et al., 2005; Sung et al., 2006; Venkataraman et al., 2005; Cormode & Muthukrishnan, 2005; Cormode & Garofalakis, 2008), databases (Selinger et al., 1979; Dasu et al., 2002; Flajolet & Nigel Martin, 1985; Lightstone, 2018; Shukla et al., 1996; Brown et al., 2005), sensor networks (Hu et al., 2022; Gaber, 2007), financial transaction data (Almeida et al., 2023), and scientific data streams (Chandrasekaran et al., 2003; Szalay et al., 2002). See (Muthukrishnan, 2005) 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 (Ben-Eliezer & Yogev, 2020; Ben-Eliezer et al., 2021; Hassidim et al., 2020; Woodruff & Zhou, 2021; Alon et al., 2021; Kaplan et al., 2021; Braverman et al., 2021; Menuhin & Naor, 2021; Attias et al., 2021; Ben-Eliezer et al., 2022; Chakrabarti et al., 2022), 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 the white-box adversarial streaming model.

[1]Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA. Correspondence to: Ying Feng <yingfeng@andrew.cmu.edu>, Aayush Jain <aayushja@andrew.cmu.edu>, David P. Woodruff <dwoodruf@cs.cmu.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1.1. White-box Adversarial Streaming Model

We consider the white-box adversarial streaming model introduced in (Ajtai et al., 2022), 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.*

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 (Ben-Eliezer et al., 2021), 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 (Selinger et al., 1979), online analytical processing (Shukla et al., 1996), data integration (Brown et al., 2005), and data warehousing (Dasu et al., 2002).

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 (Driscoll et al., 1989; Fiat & Kaplan, 2003; Kaplan, 2004).

Dynamic algorithms often consider an adaptive adversary who generates the updates upon seeing the entire data structure maintained by the algorithm during the execution (Chan, 2010; Chan & He, 2021; Roghani et al., 2022). For example, (Wajc, 2020) 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 (Szegedy et al., 2014; Goodfellow et al., 2014), with a large body of recent literature focusing on adversarial robustness of machine learning models against white-box attacks (Ilyas et al., 2018; Madry et al., 2018; Schmidt et al., 2018; Tramèr et al., 2018; Cubuk et al., 2018; Kurakin et al., 2017; Liu et al., 2017).

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 (Ajtai et al., 2022) and (Feng & Woodruff, 2023) explicitly study the white-box model for data streams and suggest the use of cryptography in designing WAR algorithms (see also (Ben-Eliezer et al., 2021) 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 the study of property preserving hashing (Boyle et al., 2019).

In this work, the central problem we consider is k-sparse recovery 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 (Ajtai et al., 2022; Feng & Woodruff, 2023). 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 (Ajtai et al., 2022; Feng & Woodruff, 2023) 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., (Berinde et al., 2008), 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.

(Feng & Woodruff, 2023) solves this problem based on the assumed hardness of the Short-Integer-Solution (SIS) problem, which is a problem that is studied in lattice-based cryptography. The robust version of the property preserving hashing (Boyle et al., 2019) can also be applied to solve the same problem. However, a notable drawback of all prior solutions proposed is their reliance on either a *random oracle* or a prohibitively long random string. This is because a key component of these algorithms is a giant, 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. Long random strings are 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., (Nisan, 1992; Indyk, 2006; Kane et al., 2010; Kacham et al., 2023). Since it was previously unknown how to generate $\mathbf{A}$ pseudorandomly while maintaining this hardness property, previous work (Feng & Woodruff, 2023) relies on a random oracle to heuristically compress such a matrix $\mathbf{A}$. This assumes that the algorithm is given read access to a long string of random bits, which is often implemented with hash-based heuristic functions such as AES or SHA256. Unfortunately, this heuristic cannot be proven secure using a standard reduction-style security proof. This motivates a central question of this work: Do there exist efficient streaming algorithms in the white-box model without a random oracle?

1.3. Our Results

Our algorithms make use of the hardness assumptions of the Learning with Error (LWE) problem and its variant, Ring LWE. Both problems are popular in cryptography known for building homomorphic encryption schemes. In this section, we state the efficiency of our algorithms under the sub-exponential hardness assumptions of these problems. However, we stress that in contrast to (Feng & Woodruff, 2023), our algorithms are robust even under weaker polynomial hardness assumptions. We let n be the dimension of the underlying input vector, and for notational convenience, we assume 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 $f \cdot \mathsf{poly}(\log n)$.

Our main result is informally stated as follows:

Theorem 1.1 (Informal)**.** *Assuming the sub-exponential hardness*[1] *of the Learning with Errors (LWE) problem, there is a WAR streaming algorithm for k-sparse recovery, which:*

- *takes $\widetilde{\mathcal{O}}(k)$ bits of space,*
- *has $\widetilde{\mathcal{O}}(1)$ update time, and*
- *has $\widetilde{\mathcal{O}}(k^{1+c})$ 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 easier oblivious streaming setting. On the other hand, assuming polynomial security of the LWE problem, there is a multiplicative overhead of n^{ϵ} for arbitrary constant $\epsilon > 0$ on our space and time complexities.

Adversarial Model in the Distributed Setting. We introduce and formalize the white-box adversarial model for distributed data processing as a means of modeling the existence of one or more malicious parties in a distributed system. We also study the sparse recovery problem in a white-box distributed model. A WAR streaming algorithm can be naively 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. In contrast to this suboptimal time implied by the naive conversion, we show that we can achieve a near-optimal processing time in a white-box distributed model assuming the *polynomial* hardness of Ring-LWE, which is a variant of the LWE problem defined over a polynomial ring. This is because Ring-LWE supports SIMD (Single-Instruction Multiple-Data) style operations that have been exploited in the design of efficient homomorphic encryption schemes in cryptography. 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 $\widetilde{\mathcal{O}}(k)$ bits of communication between each server and the coordinator,*
- *has $\widetilde{\mathcal{O}}(n)$ processing time on each server, and*
- *the coordinator spends $\widetilde{\mathcal{O}}(\max(n, k^{1+c}))$ time to output the solution given the messages from the servers, for an arbitrarily small constant $c > 0$.*

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

[1] Sub-exponential hardness means that we assume that for adversaries of size $2^{\lambda^{\beta}}$, where λ is the security parameter and $\beta > 0$ is some constant, a sample from an LWE distribution is indistinguishable from a uniformly random sample.

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 (Feng & Woodruff, 2023). The SIS assumption used in the previous work is proven to be equivalently hard to the LWE assumption we use, up to different parameters. 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	$\widetilde{\mathcal{O}}(n^c + k)$	$\widetilde{\mathcal{O}}(n)$	✗

Extensions to matrix and tensor recoveries. Similar to (Feng & Woodruff, 2023), 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. These results are given in Appendix G due to the page limit.

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

We remark that the above matrix recovery algorithm performs $\widetilde{\mathcal{O}}(nk)$ measurements using n-sparse matrices, 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 (Feng & Woodruff, 2023) 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 (Feng & Woodruff, 2023). In short, by running our k-sparse recovery algorithm with the sparsity parameter k set to $n^{1-\alpha}$ for some constant α, we can get an n^α-approximation to the L_0 norm of a vector. In comparison to (Feng & Woodruff, 2023), our construction again removes the need for a random oracle or the need to store a lot of randomness.

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 (Feng & Woodruff, 2023), we work with a suitable fully homomorphic encryption (FHE) scheme to achieve the desired property. We start with an intuitive explanation of the previous construction in (Feng & Woodruff, 2023).

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 a 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 efficiently implemented using, e.g., a random polynomial "fingerprint" (Cormode & Jowhari, 2019). However, this approach is not WAR as the randomness of the sampler is revealed and thus not independent of the stream.

Instead, (Feng & Woodruff, 2023) 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, (Feng & Woodruff, 2023) maintains a hash $h = \mathbf{A}\boldsymbol{x}$, where $\boldsymbol{x}$ is the data vector implicitly defined by the stream. 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 sub-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 λ (or even polylogarithmic assuming subexponential time assumptions).
- The hash should be computable in a streaming fashion and it should be updatable where each update takes polynomial time in the security parameter λ (or even polylogarithmic time assuming subexponential time assumptions).
- The hash key should be pseudorandom. In other words, in our real-world implementation, the white-box adversarial streaming 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 will 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 tree style hashing to hash strings of arbitrary length. Such a hash will have $\mathsf{poly}(\lambda)$-sized hashing keys. Unfortunately, the first idea suffers from a large hashing key size, which could be shortened in the random oracle (RO) model by using the 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 to leverage a suitably chosen Fully Homomorphic Encryption (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 construction bears a resemblance to the construction of a collision resistant function from a private information retrieval scheme, where the private information retrieval scheme is implemented by a fully homomorphic encryption scheme.

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 pseudorandom public 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 (Gentry et al., 2013), which satisfies the additional properties required by our construction. We remark that in addition to GSW, 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 a total complexity of $n\,\mathsf{poly}(\lambda)$, where λ is the security parameter. Therefore, we appeal to the Ring-LWE assumption, which is an algebraic variant of LWE that works over the ring $\mathbb{Z}_p[x]/(x^N + 1)$. Over this ring, we can exploit the SIMD property of Ring-LWE by packing the data as ring elements, i.e., we represent the input vectors as partitioned segments of length proportional to the security parameter (rather than the vector length n), 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 F 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. For a finite set S, we write $x \xleftarrow{\$} S$ to denote a uniform sample 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

We define the Learning with Errors problem LWE.

Definition 2.1 (Learning with Errors)**.** *Suppose we are given an integer dimension $g \in \mathbb{N}$, a modulus $q \in \mathbb{N}$, a 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 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 $\gamma = g\frac{q}{\sigma}$, which is a worst-case problem over lattices. We discuss more about the lattice problems in Appendix A.

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

Definition 2.2 (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 (Lyubashevsky et al., 2010). 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 (Smart & Vercauteren, 2011).

2.2. 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 2.1. We also define $(\mathcal{T}, \epsilon)$-*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 2.3 (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, which are elaborated in Appendix B.

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 2.4 (Pseudorandom FHE (PFHE))**.** *Let* FHE *be a fully homomorphic encryption scheme* (Definition 2.3)*. We call* FHE *a $(\mathcal{T}, \epsilon)$-pseudorandom* FHE *scheme if for all security parameters 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 following: $|\mathcal{M}| \leq \mathsf{poly}(\lambda)$. Also, there exists a $\mathsf{poly}(\lambda)$-time decoding algorithm 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* PFHE *to satisfy compactness, correctness, and security.*

There exist FHE schemes based on LWE and Ring-LWE (Gentry et al., 2013; Brakerski et al., 2012; Brakerski & Vaikuntanathan, 2011) 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 PFHE.

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 and an input vector $\boldsymbol{x} \in \mathbb{Z}^n$, output $\boldsymbol{x}$ if $\|\boldsymbol{x}\|_0 \leq k$ and report $\perp$ otherwise.*

Here and throughout, for the sparsity to make sense we always assume $k \leq n$, though 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 and an input vector $\boldsymbol{x} \in \mathbb{Z}^n$, output $\boldsymbol{x} \in \mathbb{Z}^n$ if $\|\boldsymbol{x}\|_0 \leq k$.*

As mentioned earlier, there exists a space- and time-efficient, deterministic streaming algorithm that solves the relaxed k-sparse recovery problem, which we will refer to as $\mathsf{StreamAlg}_0$ in our algorithm.

With $\mathsf{StreamAlg}_0$ and a PFHE, we are now ready to construct a WAR streaming algorithm for the k-sparse recovery problem.

Construction 3.1. Let $\mathsf{StreamAlg}_0$ be an algorithm for the Relaxed k-Sparse Recovery problem. Let PFHE be a pseudorandom FHE. Let $\lambda = \mathcal{S}(n)$ be the security parameter of PFHE, for some function $\mathcal{S}$ in n that we will specify later. Choose $q \in \mathsf{poly}(n)$ with $q \gg n \cdot N$ to be the modulus of PFHE ciphertexts, where $N \in \mathsf{poly}(n)$ is a bound on the stream length. All computations will be modulo q. 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: $C_i(\mu_1, \mu_2, \cdots, \mu_\ell) = \begin{cases} 1 & \text{if } (\mu_1, \mu_2, \cdots, \mu_\ell) \text{ is the bit representation of } i \\ 0 & \text{otherwise.} \end{cases}$

  Take $\mathcal{D}_{\widetilde{\mathsf{pk}}(\lambda)}$ to be the distribution of pseudo-public key in Definition 2.4. 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 2.4. 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 $sketch := 0$.

- $\mathsf{StreamAlg.Update}(i_t, \Delta_t)$:

  Run $\mathsf{StreamAlg}_0.\mathsf{Update}(i_t, \Delta_t)$. 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 $sketch \leftarrow sketch + \Delta_t \cdot \widehat{\mathsf{ct}_{i_t}}$.

- $\mathsf{StreamAlg.Report}()$:

  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 all $i \in [n]$. 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.

We state the white-box robustness of our construction as follows. We delay the proof to Appendix D.

Theorem 3.1 (WAR of Construction 3.1)**.** *Suppose $\mathsf{StreamAlg}_0$ is a streaming algorithm for Relaxed k-Sparse Recovery, and PFHE is a pseudorandom FHE scheme with $(\mathcal{T}, \epsilon)$-security. Let $\lambda = \mathcal{S}(n)$ be the PFHE security parameter for some function $\mathcal{S}$.*

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 function* negl.

then the StreamAlg *scheme in* Construction 3.1 *is a WAR streaming algorithm for the k-Sparse Recovery problem.*

4. Efficiency

In this section, we discuss the space and time complexity of Construction 3.1. We start by instantiating $\mathsf{StreamAlg}_0$ and PFHE used in our construction.

Deterministic Relaxed Sparse Recovery. There exist space- and time-efficient streaming algorithms for the relaxed k-sparse recovery problems. Given a sparse vector $\boldsymbol{x} \in \mathbb{Z}^n$, many of these algorithms perform some linear measurement $\Phi : \mathbb{Z}^n \to \mathbb{Z}^r$ on $\boldsymbol{x}$ with $r \ll n$ to get a compressed 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. Many 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. Instead, we use algorithms which are deterministic, such that the measurements themselves incur no space overhead.

Theorem 4.1 ((Gilbert et al., 2017))**.** *There exists a streaming algorithm* $\mathsf{StreamAlg}_0$ *for the Relaxed k-Sparse Recovery problem, 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 $c > 0$.*

Pseudorandom FHE. We focus on GSW (Gentry et al., 2013), a standard LWE based fully homomorphic encryption scheme, for our instantiation. 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 $\mathsf{LWE}_{g,h,q,\chi}$.
- $\mathsf{SecretKeyGen}(params)$: Sample $\overline{\boldsymbol{s}} \leftarrow \mathbb{Z}_q^g$. Output $\mathsf{sk} = \boldsymbol{s}^\top = [-\overline{\boldsymbol{s}}^\top \mid 1]$.
- $\mathsf{PublicKeyGen}(params, \mathsf{sk})$: 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.

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 verify in Appendix E that GSW satisfies the additional properties required by PFHE.

Efficiency Based on Subexponential LWE. We 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 $\approx_\epsilon$:

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

We set the security parameter be $\lambda = \log^r n$ for some constant $r > 1/\beta$ and $r > 1/c$. This satisfies the relationship between functions required by the Robustness Theorem 3.1: $\mathcal{T}(\log^r n) = 2^{(\log n)^{r\beta}} = \Omega(\mathsf{poly}\, n)$ 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 and Theorem 4.1 to implement $\mathsf{StreamAlg}_0$, we get the claimed result.

Corollary 4.1. *Assuming the subexponential hardness of LWE, the* $\mathsf{StreamAlg}$ *scheme in Construction 3.1 has the following complexity:*

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

In Appendix E, we prove the above corollary and show the complexity of our algorithm under different implementations of $\mathsf{StreamAlg}_0$ and hardness assumptions.

Impact statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

David P. Woodruff was supported by a Simons Investigator Award and NSF Grant No. CCF-2335412.

References

Aharonov, D. and Regev, O. Lattice problems in np conp. *J. ACM*, 52:749–765, 2005.

Ajtai, M., Braverman, V., Jayram, T., Silwal, S., Sun,

A., Woodruff, D. P., and Zhou, S. The white-box adversarial data stream model. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '22, pp. 15–27, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392600. doi: 10.1145/3517804.3526228. URL https://doi.org/10.1145/3517804.3526228.

Almeida, A., Brás, S., and Sargento, S., e. a. Time series big data: a survey on data stream frameworks, analysis and algorithms. *J Big Data*, 2023. doi: https://doi.org/10.1186/s40537-023-00760-1.

Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pp. 20–29, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi: 10.1145/237814.237823. URL https://doi.org/10.1145/237814.237823.

Alon, N., Ben-Eliezer, O., Dagan, Y., Moran, S., Naor, M., and 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.

Attias, I., Cohen, E., Shechner, M., and Stemmer, U. A framework for adversarial streaming via differential privacy and difference estimators. *CoRR*, abs/2107.14527, 2021.

Ben-Eliezer, O. and 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.

Ben-Eliezer, O., Jayaram, R., Woodruff, D. P., and Yogev, E. A framework for adversarially robust streaming algorithms. *SIGMOD Rec.*, 50(1):6–13, 2021.

Ben-Eliezer, O., Eden, T., and Onak, K. Adversarially robust streaming via dense-sparse trade-offs. In *5th Symposium on Simplicity in Algorithms, SOSA*, 2022. (to appear).

Berinde, R., Indyk, P., and 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. doi: 10.1109/ALLERTON.2008.4797556.

Boyle, E., LaVigne, R., and 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*, volume 124 of *LIPIcs*, pp. 16:1–16:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

Brakerski, Z. and 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, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-22792-9.

Brakerski, Z., Gentry, C., and 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.

Braverman, V., Hassidim, A., Matias, Y., Schain, M., Silwal, S., and 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).

Brown, P., Haas, P. J., Myllymaki, J., Pirahesh, H., Reinwald, B., and 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.

Chakrabarti, A., Ghosh, P., and Stoeckl, M. Adversarially robust coloring for graph streams. In *13th Innovations in Theoretical Computer Science Conference, ITCS*, 2022. (to appear).

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.

Chan, T. M. and 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.

Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Krishnamurthy, S., Madden, S. R., Reiss, F., and Shah, M. A. Telegraphcq: Continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pp. 668, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 158113634X. doi: 10.1145/872757.872857. URL https://doi.org/10.1145/872757.872857.

Cormode, G. and 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*, EDBT '08, pp. 745,

New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781595939265. doi: 10.1145/1353343.1353442. URL https://doi.org/10.1145/1353343.1353442.

Cormode, G. and Jowhari, H. Lp samplers and their applications: A survey. *ACM Comput. Surv.*, 52(1), feb 2019. ISSN 0360-0300. doi: 10.1145/3297715. URL https://doi.org/10.1145/3297715.

Cormode, G. and Muthukrishnan, S. What's new: finding significant differences in network data streams. *IEEE/ACM Transactions on Networking*, 13(6):1219–1232, 2005. doi: 10.1109/TNET.2005.860096.

Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018.

Dasu, T., Johnson, T., Muthukrishnan, S., and 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.

Driscoll, J. R., Sarnak, N., Sleator, D. D., and Tarjan, R. E. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

Feng, Y. and Woodruff, D. P. Improved algorithms for white-box adversarial streams. In *ICML*, 2023.

Fiat, A. and Kaplan, H. Making data structures confluently persistent. *J. Algorithms*, 48(1):16–58, 2003.

Flajolet, P. and Nigel Martin, G. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31 (2):182–209, 1985. ISSN 0022-0000. doi: https://doi.org/10.1016/0022-0000(85)90041-8. URL https://www.sciencedirect.com/science/article/pii/0022000085900418.

Forbes, M. A. and 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*, STOC '12, pp. 163–172, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312455. doi: 10.1145/2213977.2213995. URL https://doi.org/10.1145/2213977.2213995.

Gaber, M. M. *Data Stream Processing in Sensor Networks*, pp. 41–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-73679-0. doi: 10.1007/3-540-73679-4_4. URL https://doi.org/10.1007/3-540-73679-4_4.

Gentry, C., Sahai, A., and Waters, B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Canetti, R. and Garay, J. A. (eds.), *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pp. 75–92. Springer, Heidelberg, August 2013. doi: 10.1007/978-3-642-40041-4_5.

Gilbert, A. C., Li, Y., Porat, E., and Strauss, M. J. For-all sparse recovery in near-optimal time. *ACM Trans. Algorithms*, 13(3), mar 2017. ISSN 1549-6325. doi: 10.1145/3039872. URL https://doi.org/10.1145/3039872.

Goldreich, O. and Goldwasser, S. On the limits of nonapproximability of lattice problems. *J. Comput. Syst. Sci.*, 60(3):540–563, 2000. URL https://doi.org/10.1006/jcss.1999.1686.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014. URL http://arxiv.org/abs/1412.6572.

Hassidim, A., Kaplan, H., Mansour, Y., Matias, Y., and 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.

Haviv, I. and Regev, O. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In Johnson, D. S. and Feige, U. (eds.), *39th ACM STOC*, pp. 469–477. ACM Press, June 2007. doi: 10.1145/1250790.1250859.

Hu, Y., Qu, A., Wang, Y., and Work, D. B. Streaming data preprocessing via online tensor recovery for large environmental sensor networks. *ACM Trans. Knowl. Discov. Data*, 16(6), jul 2022. ISSN 1556-4681. doi: 10.1145/3532189. URL https://doi.org/10.1145/3532189.

Ilyas, A., Engstrom, L., and Madry, A. Prior convictions: Black-box adversarial attacks with bandits and priors. *CoRR*, abs/1807.07978, 2018.

Indyk, P. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53 (3):307–323, 2006.

Jafarpour, S. *Deterministic Compressed Sensing*. PhD thesis, Princeton University, 2011.

Kacham, P., Pagh, R., Thorup, M., and Woodruff, D. P. Pseudorandom hashing for space-bounded computation with applications in streaming, 2023.

Kane, D. M., Nelson, J., and 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.

Kaplan, H. Persistent data structures. In *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004.

Kaplan, H., Mansour, Y., Nissim, K., and 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.

Khot, S. Hardness of approximating the shortest vector problem in lattices. In *45th FOCS*, pp. 126–135. IEEE Computer Society Press, October 2004. doi: 10.1109/FOCS.2004.31.

Kurakin, A., Goodfellow, I. J., and Bengio, S. Adversarial machine learning at scale. In *5th International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2017.

Lall, A., Sekar, V., Ogihara, M., Xu, J., and Zhang, H. Data streaming algorithms for estimating entropy of network traffic. *SIGMETRICS Perform. Eval. Rev.*, 34 (1):145–156, jun 2006. ISSN 0163-5999. doi: 10.1145/1140103.1140295. URL https://doi.org/10.1145/1140103.1140295.

Lightstone, S. Physical database design for relational databases. In *Encyclopedia of Database Systems, Second Edition*. Springer, 2018.

Liu, Y., Chen, X., Liu, C., and Song, D. Delving into transferable adversarial examples and black-box attacks. In *5th International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2017.

Lyubashevsky, V., Peikert, C., and Regev, O. On ideal lattices and learning with errors over rings. In Gilbert, H. (ed.), *EUROCRYPT 2010*, volume 6110 of *LNCS*, pp. 1–23. Springer, Heidelberg, May / June 2010. doi: 10.1007/978-3-642-13190-5_1.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2018.

Menuhin, B. and Naor, M. Keep that card in mind: Card guessing with limited memory. *CoRR*, abs/2107.03885, 2021.

Muthukrishnan, S. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005. ISSN 1551-305X. doi: 10.1561/0400000002. URL http://dx.doi.org/10.1561/0400000002.

Nisan, N. Pseudorandom generators for space-bounded computation. *Comb.*, 12(4):449–461, 1992.

Phillips, J. M., Verbin, E., and Zhang, Q. Lower bounds for number-in-hand multiparty communication complexity, made easy, 2015.

Roghani, M., Saberi, A., and Wajc, D. Beating the folklore algorithm for dynamic matching. In *13th Innovations in Theoretical Computer Science Conference, ITCS*, 2022. (to appear).

Schmidt, L., Santurkar, S., Tsipras, D., Talwar, K., and 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.

Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., and 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.

Shoup, V. Fast construction of irreducible polynomials over finite fields. *Journal of Symbolic Computation*, 17(5):371–391, 1994. ISSN 0747-7171. doi: https://doi.org/10.1006/jsco.1994.1025. URL https://www.sciencedirect.com/science/article/pii/S074771718471025X.

Shukla, A., Deshpande, P., Naughton, J. F., and 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.

Smart, N. and Vercauteren, F. Fully homomorphic SIMD operations. Cryptology ePrint Archive, Report 2011/133, 2011. https://eprint.iacr.org/2011/133.

Sung, M., Kumar, A., Li, L., Wang, J., and 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. doi: 10.1109/ICDEW.2006.130.

Szalay, A., Gray, J., and VandenBerg, J. Petabyte scale data mining: Dream or reality? *Proceedings of SPIE - The International Society for Optical Engineering*, 08 2002. doi: 10.1117/12.461427.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. Intriguing properties of neural networks. *International Conference on Learning Representations*, 2014.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I. J., Boneh, D., and McDaniel, P. D. Ensemble adversarial training: Attacks and defenses. In *6th International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2018.

Venkataraman, S., Song, D. X., Gibbons, P. B., and Blum, A. New streaming algorithms for fast detection of superspreaders. In *Network and Distributed System Security Symposium*, 2005. URL https://api.semanticscholar.org/CorpusID:538509.

von zur Gathen, J. and Gerhard, J. *Fast polynomial evaluation and interpolation*. Cambridge University Press, 3 edition, 2013. doi: 10.1017/CBO9781139856065.013.

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.

Woodruff, D. P. and Zhang, Q. Tight bounds for distributed functional monitoring. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pp. 941–960, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312455. doi: 10.1145/2213977.2214063. URL https://doi.org/10.1145/2213977.2214063.

Woodruff, D. P. and 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.

Zhao, Q. G., Kumar, A., Wang, J., and 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. ISSN 0163-5999. doi: 10.1145/1071690.1064258. URL https://doi.org/10.1145/1071690.1064258.

Appendix A contains additional preliminaries for lattices. Appendix B contains the compactness, correctness, and security properties of FHE. In Appendix C, we recall the streaming model, the structure of a streaming algorithm, and the formal definition of white-box adversarial robustness.

We prove the robustness of our streaming K-Sparse Recovery algorithm in Appendix D and show the efficiency of the algorithm in Appendix E. Finally, in Appendix F and G, we describe our results for distributed protocols and low-rank matrix and tensor recovery, respectively.

A. Additional Lattice Preliminaries

In this section we introduce more background on the lattice problems, which helps justify our hardness assumptions based on LWE and Ring-LWE.

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.

We now define a worst-case lattice problem, the gap version of the shortest vector problem, and discuss its relation to the average-case problems (LWE and Ring-LWE) that we used.

Definition A.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 $\gamma = g^{o(1)}$ (Khot, 2004; Haviv & Regev, 2007), and is known to be inside $\mathsf{NP} \cap \mathsf{coNP}$ for $\gamma = O(\sqrt{g})$ (Aharonov & Regev, 2005; Goldreich & Goldwasser, 2000). The problem is believed to be intractable by polynomial time adversaries even for $\gamma = 2^{g^\epsilon}$ for $0 < \epsilon < 1$.

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 $\gamma = g\frac{q}{\sigma}$. On the other hand, The Ring-LWE problem is as hard as worst-case problems over ideal lattices (Lyubashevsky et al., 2010).

B. Properties of Fully Homomorphic Encryption

We elaborate the compactness, correctness, and security properties that we requires for an FHE scheme.

Definition B.1 (FHE Compactness)**.** *We say that a* FHE *scheme is compact if there exists a polynomial* $\mathsf{poly}(\cdot)$ *such that for all security parameters λ, circuits $C : \{0,1\}^\ell \to \{0,1\}$ 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 B.2 (FHE Correctness)**.** *We say that a* FHE *scheme is correct if for all security parameters 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 B.3 (FHE Security)**.** *We say that a* FHE *scheme satisfies security if for all security parameter λ, the following holds: For any PPT$_\lambda$ 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 B.1. 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 2.1 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 $\epsilon(\lambda)$.

C. Streaming Model

Notation. Let $\mathbb{1}(p)$ for some proposition p be an indicator variable, which equals 1 if p is true and 0 otherwise. 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 (Alon et al., 1996) 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 so that it takes a sublinear number of bits of space in n to represent.

We define the general framework for streaming algorithms as follows:

Definition C.1 (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* 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* 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 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 may be 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 C.2 (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 $i \in [n]$.*

D. Proof of White-box Robustness of Construction 3.1

For the concreteness of our proof, we assume that the relaxed k-sparse recovery algorithm $\mathsf{StreamAlg}_0$ that is used in our construction is deterministic and runs in polynomial time. Both conditions are satisfied by many existing sparse recovery algorithms.

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

In this section, we will show that 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 StreamAlg scheme in *Construction 3.1* is a WAR streaming algorithm that solves the k-Sparse Recovery problem.

Remark D.1. Roughly, we need $\mathsf{poly}(n) \subseteq \mathcal{T}(\mathcal{S}(n))$ to convert between the runtime of a PFHE adversary and a StreamAlg adversary. We also need $\epsilon(\mathcal{S}(n)) \leq \mathsf{negl}(n)$ to convert between the advantage of a PFHE adversary and a StreamAlg adversary.

Proof (of Theorem 3.1). 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 D.1. *Suppose* PFHE *satisfies* Definition 2.4 *and* $\mathsf{StreamAlg}_0$ *solves the Relaxed k-Sparse Recovery problem. Then 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 $\|\boldsymbol{x}\|_0 \leq k$, $\mathsf{StreamAlg}_0.\mathsf{Report}$ outputs $\boldsymbol{x}' = \boldsymbol{x}$. Since Definition 2.4 restricts $\mathsf{PFHE.Eval}$ 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, $\mathsf{StreamAlg.Report}$ correctly outputs $\boldsymbol{x}$ as desired.

□

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 C.2, where the challenger runs the $\mathsf{StreamAlg}$ 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 2.4) 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 $\mathsf{StreamAlg}$. 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.

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

Lemma D.2. *For any 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., the 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 D.3. *For any adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \geq \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]/n$.*

Before proving Lemma D.3, we make the following observation:

Claim D.1. *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 D.1, $r_t \neq f_n(\boldsymbol{x})$ 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, and 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 D.3 then follows from Claim D.1:

Proof (of Lemma D.3). Claim D.1 states that $\boldsymbol{x'} \neq \boldsymbol{x}$, and in particular, $\boldsymbol{x'}$ and $\boldsymbol{x}$ disagree on 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 $\boldsymbol{x}$ disagree. 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 D.4. *Given that* PFHE *satisfies* Definition 2.4*, 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. Also by the $(\mathcal{T}, \epsilon)$-indistinguishability in Definition 2.4, 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 D.5. *Given that* PFHE *satisfies* Definition 2.4*, 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 D.4, 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 2.4. 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 2.4 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 D.6. *Given that* PFHE *satisfies* Definition 2.4*, for any PPT$_n$ adversaries $\mathcal{A}$,* $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = 0$.

Proof. As in the proof for Claim D.1, 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 $\mathsf{StreamAlg.Report}$.

Recall the $\mathsf{StreamAlg}$ 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 $D.2$ to Lemma $D.6$:

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 StreamAlg scheme from Construction 3.1 outputs 1 with probability at most $\mathsf{negl}(n)$.

□

E. Efficiency

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

E.1. (Alternative) Deterministic Relaxed Sparse Recovery

The deterministic Relaxed k-Sparse Recovery algorithm in Theorem 4.1 achieves nearly optimal space and time. However, the measurement is "constructed" non-explicitly using the probabilistic method. For explicit construction, we consider the following result:

Theorem E.1 ((Jafarpour, 2011))**.** *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 (amortized) $\widetilde{\mathcal{O}}(1)$.*
- $\mathsf{StreamAlg}_0.\mathsf{Report}$ *runs in $\widetilde{\mathcal{O}}(k^2)$.*

The above theorem except the amortized runtime has been shown in (Jafarpour, 2011).

Proof (of the runtime in Theorem E.1*).* The measurement Φ in Theorem $E.1$ is a $2k \times n$ Vandermonde matrix explicitly defined [2] 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 structured. 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. We need the following result for fast multi-point evaluation of polynomials and the Transposition Principle:

Theorem E.2 ((von zur Gathen & Gerhard, 2013))**.** *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.*

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

Theorem E.3 (Transposition Principle, (Shoup, 1994))**.** *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 $E.2$ and Theorem $E.3$ together show the following update time for Vandermonde measurements:

Proposition E.1. *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 (of Proposition E.1). 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 $\mathsf{StreamAlg}_0.\mathsf{Update}$.

□

E.2. GSW Satisfies PFHE Properties

We 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:

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

- ($\mathbb{Z}$ 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 the normal GSW decryption for integer messages. For completeness, we briefly describe 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).

E.3. Efficiency of PFHE Based on Subexponential LWE

We use GSW to implement PFHE and set the security parameter to be $\lambda = \log^r n$ for some constant $r > 1/\beta$ and $r > 1/c$. We get the following proposition.

Proposition E.2. *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* 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 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. $\widetilde{\mathsf{pk}}$, $sketch$, 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 $sketch \leftarrow sketch + \Delta_t \cdot \widehat{\mathsf{ct}_{i_t}}$.

   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 na¨

   $velyimplementedusing$O$(\log n)$ arithmetic operations, each of which runs in 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 $hash \leftarrow \sum_{i\in[n]} \widehat{\mathsf{ct}_i}\cdot \boldsymbol{x}'_i$.

   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 $\widetilde{\mathcal{O}}(k)$.

□

E.4. Efficiency of PFHE 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 E.3. *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 E.2, except that now the dimension of the ciphertext matrices blows up from $\mathsf{poly}\log n$ to n^w for an arbitrarily small $w > 0$.

□

E.5. Complexity of Construction 3.1

Corollary 4.1 that we include in the main body follows from Theorem 4.1 and Proposition E.2. 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 3.1, given different instantiations of $\mathsf{StreamAlg}_0$ and PFHE. c denotes an arbitrarily small positive constant.

RELAXED REC. SCHEME	LWE ASSUMPTION	SPACE	UPDATE TIME	REPORT TIME
THEOREM 4.1	SUBEXP	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(1)$	$\widetilde{\mathcal{O}}(k^{1+c})$
THEOREM E.1	SUBEXP	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(1)$	$\widetilde{\mathcal{O}}(k^2)$
THEOREM 4.1	POLY	$\widetilde{\mathcal{O}}(k+n^c)$	$\widetilde{\mathcal{O}}(n^c)$	$\widetilde{\mathcal{O}}(k \cdot (k^c+n^c))$
THEOREM E.1	POLY	$\widetilde{\mathcal{O}}(k+n^c)$	$\widetilde{\mathcal{O}}(n^c)$	$\widetilde{\mathcal{O}}(k \cdot (k+n^c))$

F. Distributed K-Sparse Recovery from Ring LWE

F.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., (Phillips et al., 2015)), 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 the 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 (Woodruff & Zhang, 2012) and the references therein on the distributed model.

Similar to the streaming model, we focus on designing a robust protocol in a white-box adversarial distributed setting, where the dataset is generated and partitioned across 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 F.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* DistProt *that computes $\mathcal{F}$ contains a tuple of algorithms* DistProt = (DistProt.Setup, DistProt.Server, 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 F.2. *We say that a distributed protocol* 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 F.1. 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 StreamAlg.Update on the same initial state $\zeta \leftarrow$ 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 linear, the coordinator can run 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 a reduction if StreamAlg.Update and StreamAlg.Report are deterministic. (Otherwise, the 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 monitor the past randomness already used by StreamAlg.) Our Construction 3.1 satisfies this condition.

F.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 F.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 an arbitrary length-n input vector, as described in the previous instantiation Section E.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.

F.3. Efficiency

In this section, we focus on showing that we can use BV (Brakerski & Vaikuntanathan, 2011), a Ring-LWE based FHE scheme, to achieve near-linear processing time on DistProt.Server under the *polynomial* ring-LWE assumption.

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

Proposition F.1. *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 F.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 reporting time of StreamAlg in Construction 3.1 (except for 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 (Brakerski & Vaikuntanathan, 2011) 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

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

where s is sampled from the noise distribution χ and a_i are uniform in R_q.

We have the following complexity:

Proposition F.2. *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 F.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 F.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 $\mathcal{O}(n^{1-w} \cdot n^w \operatorname{poly} \log(n^w)) = \widetilde{\mathcal{O}}(n)$, since two ring elements can be multiplied in $\mathcal{O}(n^w \operatorname{poly} \log(n^w))$ time using the 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 F.1 remains correct while achieving near-linear runtime of DistProt.Coordinator and DistProt.Server.

□

G. Low-Rank Matrix and Tensor Recovery

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

Definition G.1 (Rank-k Matrix Recovery)**.** *Given a rank parameter k and an input matrix $\mathbf{X} \in \mathbb{Z}^{n \times m}$, output $\mathbf{X}$ if $rank(\mathbf{X}) \leq k$ and report $\perp$ otherwise.*

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

Definition G.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 G.3 (Rank-k Tensor Recovery)**.** *Given a rank parameter k and an input tensor $\mathbf{X} \in \mathbb{Z}^{n^d}$, output $\mathbf{X}$ if $CP - rank(\mathbf{X}) \leq k$ and report $\perp$ otherwise.*

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 (Feng & Woodruff, 2023), 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 (Forbes & Shpilka, 2012). 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 G.1. *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 (Forbes & Shpilka, 2012). 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 F.

Proposition G.2. *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 number of bits of communication between each server and the coordinator equals the size of the state in the above streaming algorithms. For 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 $\widetilde{\mathcal{O}}(n^2k)$ processing time at each server. □

We remark that the rank-k matrix recovery algorithm in the previous work (Feng & Woodruff, 2023) requires $\widetilde{\mathcal{O}}(n^3k)$ processing time in the distributed setting, which our construction improves upon.