How to Compress Encrypted Data

Nils Fleischhacker[1⋆] and Kasper Green Larsen[2⋆⋆] Mark Simkin[3⋆⋆⋆]

[1] Ruhr University Bochum
[2] Aarhus University
[3] Ethereum Foundation

Abstract. We study the task of obliviously compressing a vector comprised of n ciphertexts of size ξ bits each, where at most t of the corresponding plaintexts are non-zero. This problem commonly features in applications involving encrypted outsourced storages, such as searchable encryption or oblivious message retrieval. We present two new algorithms with provable worst-case guarantees, solving this problem by using only homomorphic additions and multiplications by constants. Both of our new constructions improve upon the state of the art asymptotically and concretely.
Our first construction, based on sparse polynomials, is perfectly correct and the first to achieve an asymptotically optimal dependency on t in the compression rate by compressing the input vector into $\mathcal{O}(t\xi)$ bits. Compression can be performed homomorphically by performing $\mathcal{O}(n \log n)$ homomorphic additions and multiplications by constants. The main drawback of this construction is a decoding complexity of $\Omega(\sqrt{n})$.
Our second construction is based on a novel variant of invertible bloom lookup tables and is correct with probability $1 - 2^{-\kappa}$. It has a slightly worse compression rate compared to our first construction as it compresses the input vector into $\mathcal{O}(\xi\kappa t/\log t)$ bits, where $\kappa \geq \log t$. In exchange, both compression and decompression of this construction are highly efficient. The compression complexity is dominated by $\mathcal{O}(n\kappa)$ homomorphic additions and multiplications by constants. The decompression complexity is dominated by $\mathcal{O}(\xi\kappa t/\log t)$ decryption operations and equally many inversions of a pseudorandom permutation.

1 Introduction

It is well known that in general encrypted data cannot be compressed. In this work, we study the task of compressing encrypted data, when a small amount of knowledge about the structure of the underlying plaintexts is known. More concretely, we consider encryptions of vectors $\boldsymbol{m} = (m_1, \ldots, m_n)$ where at most t distinct coordinates in $\boldsymbol{m}$ are non-zero. In the context of outsourced storage applications the task of compressing such vectors appears naturally.

In searchable encryption [SWP00, BDOP04], we have a client Charlie, who holds a vector $(m_1, \ldots, m_n)$ of data elements and wants to store it remotely on server Sally. To hide the contents of the data elements, Charlie encrypts the data vector under a secret key only she knows before sending it to Sally. Later on, Charlie may want to search through the vector and retrieve all elements that match some secret keyword. A series of recent works [YSK+13, LLN15, CKK16, CKL15, AFS18, CDG+21] have shown how to construct searchable encryption schemes from fully homomorphic encryption [RAD78, Gen09] with reasonable concrete efficiency. Conceptually, these approaches are comprised of two major steps. First Charlie sends a short keyword-dependent hint to the server, who uses it to obliviously transform the vector of ciphertexts $(c_1, \ldots, c_n)$ into a new vector $\tilde{\boldsymbol{c}} = (\tilde{c}_1, \ldots, \tilde{c}_n)$, where for $i \in \{1, \ldots, n\}$ the ciphertext $\tilde{c}_i$ is either an encryption of the original message m_i in c_i or zero, depending on whether m_i was matching the keyword of Charlie or not. In the second step, the server obliviously compresses the vector $\tilde{\boldsymbol{c}}$ under the assumption that no more than t ciphertexts

[⋆] mail@nilsfleischhacker.de. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.
[⋆⋆] larsen@cs.au.dk. Supported by Independent Research Fund Denmark (DFF) Sapere Aude Research Leader grant No 9064-00068B.
[⋆⋆⋆] mark.simkin@ethereum.org

were matching the keyword and sends it back to Charlie. If the assumption about the sparsity of the vector $\tilde{\boldsymbol{c}}$ was correct, then Charlie successfully decodes the vector and obtains the desired result. If the assumption was not correct, then Charlie may not be able to retrieve the output.

The best compression algorithm used in this context is due to Choi et al. [CDG+21], which compresses $\tilde{\boldsymbol{c}}$ into a bit string of length $\Omega(t\xi(\kappa + \log n))$, where ξ is the length of one ciphertext entry. Under the assumption that the plaintext messages are of a specific form[4], the authors show that Charlie can correctly decode the vector with probability $1 - 2^{-\kappa}$. Both compressing and decoding are computationally concretely efficient.

In the oblivious message retrieval setting, recently introduced by Liu and Tromer [LT22], we have a server Sally, who keeps a public bulletin board, and multiple clients Charlie, Chucky, and Chris. Each of the clients can post encrypted messages for any of the other clients on the bulletin board, but would like to hide who is the recipient of which message. At some point, for example, Chucky may want to retrieve all messages that are intended for him. Naively, he could simply download all contents from Sally's bulletin board, but this would incur a large bandwidth overhead that is linear in the total number of messages stored by Sally. Instead, the idea behind oblivious message retrieval is to let Chucky generate a short identity-dependent hint that can be used by Sally to obliviously generate a short message that contains all relevant encrypted messages for Chucky. Conceptually, the construction of Liu and Tromer follows the exact same blueprint as the searchable encryption scheme outlined above. First Sally obliviously filters her vector with the hint provided by Chucky and then she obliviously compresses the filtered vector under the assumption that not too many messages are addressed to Chucky.

From an efficiency perspective, the solution of Liu and Tromer is rather expensive. To compress, Sally performs $\Omega(tn + \kappa t \log t)$ homomorphic additions and $\Omega(tn)$ homomorphic multiplications by constants, where κ is the correctness error defined as in the searchable encryption example. Sally's message to Chucky is $\Omega(\xi t + \xi \kappa t \log t)$ bits long. To decode the result from Sally's message, Chucky needs to perform gaussian elimination on a matrix of size $\mathcal{O}(t) \times \mathcal{O}(t)$, which incurs a computational overhead of $\Omega(t^3)$. The authors provide heuristic optimizations of their constructions that improve their performance significantly, but unfortunately these come without asymptotic bounds or provable correctness guarantees.

Taking a step back and looking at the two applications described above from a more abstract point of view, one can recognize that both follow a very similar blueprint. In the first step, both apply some vastly different techniques to convert a vector of ciphertexts into a sparse vector containing only the desired entries. In the second step, both works solve the identical problem. They both need to compress a sparse homomorphically encrypted vector with nothing but the knowledge of how many entries are non-zero, and in particular without any knowledge about which entries are zero and which ones are not. How to compress such a sparse encrypted vector is the topic of this work.

1.1 Our Contribution

We present two new algorithms, one based on polynomials and one based on algorithmic hashing, for compressing sparse encrypted vectors, which both improve upon the prior state of the art in terms of compression rate. Our algorithms only rely on homomorphic additions and homomorphic multiplications by constants. Both of our constructions have provable worst-case bounds for all their parameters.

Compressing via Polynomials. Our first construction (Section 4) is perfectly correct and is based on the concept of sparse polynomial interpolation. Its compression rate has an asymptotically optimal dependence on t, as the compressed vector is merely $\mathcal{O}(\xi \cdot t)$ bits large. During compression one needs to perform $\mathcal{O}(n \log n)$ homomorphic additions and equally many homomorphic multiplications by constants. The main bottleneck of this solution is the decompression complexity of $\tilde{\mathcal{O}}(t \cdot \sqrt{n})$, which depends on the length n of the original vector. Although the compression rate is much better than that of previous works, such as those Choi et al. [CDG+21] and that of Liu and Tromer [LT22], this construction suffers from a slower decompression time.

[4] This assumption can be removed at the cost of doubling the size of the compressed vector and additionally assuming that one is not only given $\tilde{\boldsymbol{c}}$, but also some auxiliary vector $\hat{\boldsymbol{c}}$ as the output of the first step of their protocol.

Compressing via Hashing. Our second construction (Section 5) is a randomized hashing based solution, which is correct with probability $1 - 2^{-\kappa}$, where the probability is taken over the random coins of the compression algorithm. We develop a novel data structure that is heavily inspired by the invertible bloom lookup tables of Goodrich and Mitzenmacher [GM11], but can be applied efficiently to encrypted data. Both compression and decompression are highly efficient. During compression one needs to perform $\mathcal{O}(n\kappa/\log t)$ homomorphic additions and multiplications by constants, where $\kappa \geq \log t$. During decompression the main costs come from $\mathcal{O}(\kappa t/\log t)$ many decryptions and equally many evaluations of a pseudorandom permutation. In contrast to our polynomial based solution, however, the compressed vector is $\mathcal{O}(\xi\kappa t/\log t)$ bits large. Nevertheless, this construction outperforms all prior works in terms of compression rate, while having either superior or comparable compression and decompression complexities.

1.2 Strawman Approach

When the sparse vector is encrypted using a fully homomorphic encryption scheme, conceptually simple solutions to the compression problem exist. For instance, Sally could just homomorphically sort all entries in the vector and then only send back the t largest entries. Such solutions, however, require her to perform multiplications of encrypted values. This is problematic for multiple reasons. Multiplications of encrypted values are much more computationally expensive than homomorphic additions or multiplications by constants. Since Sally may potentially store a very large database, we would like to minimize her computational overhead. Furthermore, if the data is encrypted using a somewhat homomorphic encryption scheme, then the multiplicative depth of the circuit that can be executed on the vector by Sally is bounded. Ideally, we would like the compression step to be concretely efficient and not require the use of any multiplications of encrypted values. For these reasons, we only focus on compression algorithms that require homomorphic additions and multiplications by constants in this work.

2 Preliminaries

Notation. Given a possibly randomized function $f : X \to Y$, we will sometimes abuse notation and write $f(\boldsymbol{x}) := (f(x_1), \ldots, f(x_n))$ for $\boldsymbol{x} \in X^n$. For a set X, we write $x \leftarrow X$ to denote the process of sampling a uniformly random element $x \in X$. For a vector $\boldsymbol{v} \in X^n$, we write v_i to denote its i-th component. For a matrix $M \in X^{n \times m}$, we write $M[i, j]$ to denote the cell in the i-th row and j-th column. We write $[n]$ to denote the set $\{1, \ldots, n\}$. For a set X^n, we define the scissor operator $✄(X^n) := \{(x_1, \ldots, x_n) \in X^n \mid x_i \neq x_j \ \forall i, j \in [n]\}$ to denote the subset of X^n consisting only of those vectors with unique entries.

Definition 1 (Sparse Vector Representation). *Let $\mathbb{F}_q$ be a field and let $\boldsymbol{a} \in \mathbb{F}_q^n$ be a vector. The sparse representation of $\boldsymbol{a}$ is the set* $\mathsf{sparse}(\boldsymbol{a}) := \{(i, a_i) \mid a_i \neq 0\}$.

2.1 Homomorphic Encryption

Informally, a homomorphic encryption scheme allows for computing an encryption of $f(\boldsymbol{m})$, when only given the description of f and an encryption of message vector $\boldsymbol{m}$. Throughout the paper, we assume that functions are represented as circuits composed of addition and multiplication gates.

Definition 2. *A homomorphic encryption scheme $\mathcal{E}$ is defined by a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *that work as follows:*

$\mathsf{Gen}(1^\lambda)$**:** *The key generation algorithm takes the security parameter 1^λ as input and returns a secret key* sk *and public key* pk. *The public key implicitly defines a message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. We denote the set of all public keys as $\mathcal{P}$.*

$\mathsf{Enc}(\mathsf{pk}, m)$**:** *The encryption algorithm takes the public key* pk *and message $m \in \mathcal{M}$ as input and returns a ciphertext $c \in \mathcal{C}$.*

$\mathsf{Eval}(\mathsf{pk}, f, \boldsymbol{c})$**:** *The evaluation algorithm takes the public key* pk*, a function* $f : \mathcal{M}^n \to \mathcal{M}^m$*, and a vector* $\boldsymbol{c} \in \mathcal{C}^n$ *of ciphertexts as input and returns a new vector of ciphertexts* $\tilde{\boldsymbol{c}} \in \mathcal{C}^m$*.*

$\mathsf{Dec}(\mathsf{sk}, c)$**:** *The deterministic decryption algorithm takes the secret key* sk *and ciphertext* $c \in \mathcal{C}$ *as input and returns a message* $m \in \mathcal{M} \cup \{\bot\}$*.*

Throughout the paper we assume that the ciphertext size is fixed and does not increase through the use of the homomorphic evaluation algorithm. We extend the definition of Enc and Dec to vectors and matrices of messages and ciphertexts respectively, by applying them componentwise, i.e., for any matrix $\boldsymbol{M} \in \mathcal{M}^{n\times m}$, we have $\mathsf{Enc}(\mathsf{pk}, \boldsymbol{M}) = \boldsymbol{C}$ with $\boldsymbol{C} \in \mathcal{C}^{n\times m}$ and $C[i,j] = \mathsf{Enc}(\mathsf{pk}, M[i,j])$ and equivalently $\mathsf{Dec}(\mathsf{sk}, \boldsymbol{C}) = \boldsymbol{M}'$ with $\boldsymbol{M}' \in \mathcal{M}^{n\times m}$ and $M'[i,j] = \mathsf{Dec}(\mathsf{sk}, C[i,j])$. Let $\mathcal{E}$ be an additively homomorphic encryption scheme with message space $\mathcal{M} = \mathbb{F}_q$ for some prime power q. And let $f : \mathbb{F}_q^2 \to \mathbb{F}_q$, $f(a,b) := a + b$ and let $g_\alpha : \mathbb{F}_q \to \mathbb{F}_q$, $g(a) := \alpha \cdot a$ for any constant $\alpha \in \mathbb{F}_q$. For notational convenience we write $\mathsf{Eval}(\mathsf{pk}, f, (c_1, c_2)^\intercal)$ as $c_1 \boxplus c_2$ and $\mathsf{Eval}(\mathsf{pk}, g_\alpha, c)$ as $\alpha \boxdot c$ with pk being inferrable from context. For the sake of simplicity we restrict ourselves to homomorphic encryption schemes with unique secret keys, i.e. for a given pk, there exists at most one sk, such that $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$. We write $\mathsf{Gen}^{-1}(\mathsf{pk})$ to denote the – not efficiently computable – unique secret key.

Later on in the paper, it will be convenient for us to talk about ciphertexts that may not be fresh encryptions, but still allow for some homomorphic operations to be performed on them.

Definition 3 ($\mathcal{Z}$-Validity). *Let* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *be a homomorphic encryption scheme, let* $\mathcal{Z}$ *be a class of circuits, and let* pk *be a public key. A vector* $\boldsymbol{c}$ *of ciphertexts is* $\mathcal{Z}$*-valid for* pk*, iff for all functions* $f \in \mathcal{Z}$ *it holds that* $\bot \notin \mathsf{Dec}(\mathsf{Gen}^{-1}(\mathsf{pk}), \boldsymbol{c})$ *and* $\mathsf{Dec}(\mathsf{Gen}^{-1}(\mathsf{pk}), \mathsf{Eval}(\mathsf{pk}, f, \boldsymbol{c}) = f(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))$*. We denote by* $\mathsf{vld}(\mathcal{Z}, \mathsf{pk})$ *the set of ciphertext vectors* $\mathcal{Z}$*-valid for* pk*.*

2.2 Polynomial Kung Fu

Let $f(x) = \sum_{i=0}^{d} a_i \cdot x^i \in \mathbb{F}_q[x]$ be a polynomial with coefficients from a finite field $\mathbb{F}_q$. The degree of f is defined as the largest exponent in any monomial with a non-zero coefficient. We say that f is s-sparse, if the number of non-zero monomials is at most s or more formally if $|\{a_i \mid a_i \neq 0 \wedge i \in [n]\}| \leq s$. It is well-known that any polynomial of degree at most d can be interpolated from $d+1$ evaluation points. In this work, we will make use of the less well-known fact that sparse polynomials can be interpolated from a number of evaluation points that is linear in the polynomial's sparsity. The first algorithms for interpolating sparse univariate and multivariate polynomials were presented by Prony [dP95] and Ben-Or and Tiwari [BT88] respectively. We will make use of the following result by Huang and Gao [HG19] for sparse interpolation of univariate polynomials over finite fields:

Theorem 1 ([HG19]). *Let* $f \in \mathbb{F}_q[x]$ *be a* s*-sparse univariate polynomial of degree at most* d *with coefficients from a finite field* $\mathbb{F}_q$*. Let* $\omega \in \mathbb{F}_q$ *be a primitive* $2(s+1)$*-th root of unity. There exists an algorithm* $\mathsf{Interpolate}$ *that takes evaluations* $f(\omega^0), \ldots, f(\omega^{2s+1})$ *as input and returns the coefficients of* f *in sparse representation in time* $\tilde{\mathcal{O}}(s \cdot \sqrt{d})$*.*

The algorithm of Huang and Gao relies on a subroutine for finding discrete logarithms. Using Shank's algorithm [Sha71] for this step, we obtain the computational complexity stated in the above theorem with deterministic performance guarantees.

Another tool we will use is the Fast Fourier Transform, (re-)discovered by Cooley and Tukey [CT65] which allows for evaluating a degree d polynomial given as a list of coefficients at $\ell \leq d$ evaluation points simultaneously in time $\mathcal{O}(d \log d)$. More precisely, for a fixed set of evaluation points $(\omega^0, \ldots, \omega^\ell)$, one can represent the circuit taking the polynomial coefficients $(a_0, \ldots, a_d)$ as input and returning $(f(\omega^0), \ldots, f(\omega^\ell))$ as a series of $\mathcal{O}(\log d)$ alternating layers of $\mathcal{O}(d)$ addition or multiplication by constants gates respectively.

Theorem 2 (Fast Fourier Transform). *Let* $d, \ell \in \mathbb{N}$ *with* $d \geq \ell$*. Let* $f = \sum_{i=0}^{d} a_i \cdot x^i \in \mathbb{F}_q[x]$ *be a polynomial of degree at most* d *with coefficients from a finite field* $\mathbb{F}_q$*. Let* $\omega \in \mathbb{F}_q$ *be a primitive* ℓ*-th root of unity. There exists an arithmetic circuit* FFT *comprised of a series of* $\mathcal{O}(\log d)$ *alternating layers of* $\mathcal{O}(d)$ *addition or multiplication by constants gates respectively that takes* $(a_0, \ldots, a_d)$ *as well as* $(\omega^0, \ldots, \omega^\ell)$ *as input and returns* $(f(\omega^0), \ldots, f(\omega^\ell))$*.*

2.3 Invertible Bloom Lookup Tables

An invertible Bloom lookup table (IBLT) is a data structure first introduced by Goodrich and Mitzenmacher [GM11] that supports three operations called Insert, Peel, and List. The insertion operations adds elements to the data structure, the deletion operations removes them[5] and the list operation recovers all currently present elements with high probability, if not too many elements are present.

The data structure consists of two $\gamma \times 2t$ matrices C, the count matrix and V the valueSum matrix. It further requires t-wise independent hash functions $h_i : \{0,1\}^* \to [2t]$ for $i \in [\gamma]$. Initially all values are set to 0. To insert an element x into the data structure, we locate the cells $C_{i,h_i(x)}$ and $V_{i,h_i(x)}$ for $i \in [\gamma]$ and add 1 to each counter and x to each valueSum. To remove an element, we perform the inverse operations. To list all elements currently present in (C, V), we repeatedly perform a peeling operation until (C, V) is empty. The peeling operation finds a cell with counter 1, adds that corresponding valueSum value to the output list and deletes the element from (C, V). The only way the list operation may fail is if (C, V) is not empty, but the peeling operation cannot find any cell with counter 1. It has been shown by Goodrich and Mitzenmacher that this probability decreases exponentially in $\gamma \log t$. We formally describe the algorithms in Figure 1.

```
Insert$(B, X)$
$\boldsymbol{V} := 0^{\gamma\times 2t}$
$\boldsymbol{C} := 0^{\gamma\times 2t}$
foreach $(i, x) \in [\gamma] \times X$ do
    $j := h_i(x)$
    $\boldsymbol{V}[i, j] := \boldsymbol{V}[i, j] + m_d$
    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] + 1$
return $(\boldsymbol{C}, \boldsymbol{V})$
```

```
List$((\boldsymbol{C}, \boldsymbol{V}))$
$S := \emptyset$
while $\exists\, (i^*, j^*) \in [\gamma] \times [2t].\ \boldsymbol{C}[i^*, j^*] = 1$ do
    $m := \boldsymbol{M}'[i^*, j^*]$
    $S := S \cup \{m\}$
$(\boldsymbol{C}, \boldsymbol{V}) :=$ Peel$((\boldsymbol{C}, \boldsymbol{V}), m)$
return $S$
```

```
Peel$((\boldsymbol{C}, \boldsymbol{V}), m)$
foreach $i \in [\gamma]$
    $j := h_i(m)$
    $\boldsymbol{V}'[i, j] := \boldsymbol{V}'[i, j] - m$
    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] - 1$
return $(\boldsymbol{C}, \boldsymbol{V})$
```

Fig. 1. An invertible Bloom lookup table

Theorem 3 **([GM11]).** *Let $h_1, \ldots, h_\gamma$ be t-wise independent hash functions, then for any $X = \{x_1, \ldots, x_t\}$ it holds that*

$$\Pr\left[B := \mathsf{Insert}((0^2)^{\gamma\times 2t}, X) : \mathsf{List}(B) \neq X\right] \leq \mathcal{O}\left(2^{-\gamma \log t}\right),$$

where the probability is taken over the random choices of $h_1, \ldots, h_\gamma$.

Remark 1. The construction of an IBLT can be modified to store tuples of values, by maintaining multiple valueSum matrices, one for each component. As long as one of the components remains unique among all inserted values, it is sufficient to use this component as input to the has functions, without affecting Theorem 3. We will make use of this in our construction in Section 5.

3 Ciphertext Compression

In this section we formally define the concept of a ciphertext compression scheme (Compress, Decompress). Intuitively, the compression algorithm takes the public encryption key pk as well as a vector of ciphertexts $\boldsymbol{c}$ from some family $\mathcal{F}_{\mathsf{pk}}$ of ciphertext vectors as input and returns some compressed representation thereof. The decompression algorithm gets the compressed representation as well as the secret decryption key as input and should return the decryption of $\boldsymbol{c}$.

[5] For the present discussion, we assume that only previously inserted elements are deleted.

Definition 4 (Ciphertext Compression Scheme). *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Enc}, \mathsf{Dec})$ be a homomorphic public key encryption scheme with ciphertext size $\xi = \xi(\lambda)$. Let $\mathcal{P}$ be the public key space of $\mathcal{E}$. For each $\mathsf{pk} \in \mathcal{P}$ let $\mathcal{F}_{\mathsf{pk}}$ be a set of ciphertext vectors. A δ-compressing, $(1-\epsilon)$-correct ciphertext compression scheme for the family $\mathcal{F} := \{\mathcal{F}_{\mathsf{pk}} \mid \mathsf{pk} \in \mathcal{P}\}$ is a pair of PPT algorithms $(\mathsf{Compress}, \mathsf{Decompress})$, such that for any $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and any $\boldsymbol{c} \in \mathcal{F}_{\mathsf{pk}}$ the output length of $\mathsf{Compress}(\mathsf{pk}, \boldsymbol{c})$ is at most $\delta\xi|\boldsymbol{c}|$ and it holds that*

$$\Pr[\mathsf{Decompress}(\mathsf{sk}, \mathsf{Compress}(\mathsf{pk}, \boldsymbol{c})) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))] = 1 - \epsilon(\lambda),$$

where the probability is taken over the random coins of the compression and decompression algorithms.

Remark 2. Note, that a ciphertext compression scheme gives no guarantee whatsoever in the case where $\boldsymbol{c} \notin \mathcal{F}_{\mathsf{pk}}$.

4 Compression via Sparse Polynomials

In this section we present our first construction, which is based on the idea of interpolating sparse polynomials. Given the right building blocks, the construction is conceptually very simple. We simply view the sparse encrypted vector $(c_1, \ldots, c_n)$ as the coefficient representation of sparse polynomial. Using the Fast Fourier Transform, we homomorphically evaluate this polynomial efficiently at some sufficient number of points. These encrypted evaluations will constitute the compression of the vector. To obtain the original vector during decompression, we simply decrypt the evaluation points and interpolate the corresponding sparse polynomial.

Definition 5 (Fast Fourier Functions). *The class of fast fourier functions is the set of functions $\mathcal{Z}^{\ell}_{\mathsf{FFT}} = \{f^{\ell}_{\boldsymbol{a}} \mid \boldsymbol{a} \in \mathbb{F}_q^n\}$ with*

$$f^{\ell}_{\boldsymbol{a}} : \mathbb{F}_q^{\ell} \to \mathbb{F}_q^{\ell}, \quad f_{\boldsymbol{a}}(\boldsymbol{x}) := \mathsf{FFT}(\boldsymbol{a}, \boldsymbol{x}).$$

Definition 6 ($\mathcal{Z}^{2(t+1)}_{\mathsf{FFT}}$-Valid Low Hamming Weight Ciphertext Vectors). *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Enc}, \mathsf{Dec})$ be a homomorphic public key encryption scheme. For any $\mathsf{pk} \in \mathcal{P}$, let*

$$\mathcal{F}^{\mathsf{FFT}}_{t,\mathsf{pk}} := \left\{\boldsymbol{c} \in \mathsf{vld}(\mathcal{Z}^{2(t+1)}_{\mathsf{FFT}}, \mathsf{pk}) \mid \mathsf{hw}(\mathsf{Dec}(\mathsf{Gen}^{-1}(\mathsf{pk}), \boldsymbol{c})) < t\right\}.$$

We then define the family of $\mathcal{Z}^{2(t+1)}_{\mathsf{FFT}}$-valid ciphertext vectors with low hamming weight as $\mathcal{F}^{\mathsf{FFT}}_t := \{\mathcal{F}^{\mathsf{FFT}}_{t,\mathsf{pk}} \mid \mathsf{pk} \in \mathcal{P}\}$.

```
Compress(pk, $\boldsymbol{c}$)
    $\tilde{\boldsymbol{c}} \leftarrow \mathsf{Eval}\left(\mathsf{pk}, \mathsf{FFT}\left(\cdot, (\omega^0, \ldots, \omega^{2t+1})\right), \boldsymbol{c}\right)$
    return $\tilde{\boldsymbol{c}}$
```

```
Decompress(sk, $\tilde{\boldsymbol{c}}$)
    $\boldsymbol{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, \tilde{\boldsymbol{c}})$
    $S \leftarrow \mathsf{Interpolate}(\boldsymbol{m})$
    return $S$
```

Fig. 2. A ciphertext compression scheme for $\mathcal{F}^{\mathsf{FFT}}_t$ based on sparse polynomials.

Theorem 4. *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a IND-CPA secure additively homomorphic encryption scheme with message space $\mathcal{M} = \mathbb{F}_q$ with ciphertext size $\xi = \xi(\lambda)$ and let $\omega \in \mathbb{F}_q$ be a $2(t+1)$-th primitive root of unity. Then $(\mathsf{Compress}, \mathsf{Decompress})$ from Figure 2 is a $2(t+1)/n$-compressing perfectly correct ciphertext compression scheme for vectors of $\mathcal{Z}^{2(t+1)}_{\mathsf{FFT}}$-valid ciphertexts.*

Proof. Let $\boldsymbol{c}$ be an arbitrary, but fixed $\mathcal{Z}_{\mathsf{FFT}}^{2(t+1)}$-valid ciphertext vector and let S be an arbitrary vector in sparse representation. Due to the validity condition on $\boldsymbol{c}$ we know that

$$\Pr\left[\begin{array}{r}\tilde{\boldsymbol{c}} \leftarrow \mathsf{Eval}\left(\mathsf{pk}, \mathsf{FFT}\left(\cdot,(\omega^0,\dots,\omega^{2t+1})\right),\boldsymbol{c}\right)\\ \boldsymbol{m} \leftarrow \mathsf{Dec}(\mathsf{sk},\tilde{\boldsymbol{c}})\\ S = \mathsf{Interpolate}(\boldsymbol{m})\end{array}\right] = \Pr\left[\begin{array}{r}\boldsymbol{m} \leftarrow \mathsf{Dec}(\mathsf{sk},\boldsymbol{c})\\ \boldsymbol{m'} \leftarrow \mathsf{FFT}\left(\boldsymbol{m},(\omega^0,\dots,\omega^{2t+1})\right)\\ S = \mathsf{Interpolate}(\boldsymbol{m'})\end{array}\right].$$

Furthermore, the $\mathcal{Z}_{\mathsf{FFT}}^{2(t+1)}$-validity of $\boldsymbol{c}$ tells us that $\mathsf{Dec}(\mathsf{sk},\boldsymbol{c})$ is a vector of hamming weight at most t or, when viewed as a polynomial in coefficient representation, a t-sparse polynomial of degree at most n. From Theorem 1 it follows this sparse polynomial can be correctly interpolated from its $2(t+1)$ evaluations produced by $\mathsf{FFT}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}),(\omega^0,\dots,\omega^{2t+1}))$ and therefore

$$\mathsf{Interpolate}(\mathsf{FFT}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}),(\omega^0,\dots,\omega^{2t+1}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c})).$$

The output of $\mathsf{Compress}$ is a vector of $2(t+1)$ ciphertexts of size ξ and thus the scheme is $2(t+1)/n$ compressing.

5 Compression via IBLTs

In this section we present our second construction, which is on a variant of invertible Bloom lookup tables. Given a vector of ciphertexts $\boldsymbol{c}$ the idea is to homomorphically insert the corresponding non-zero plaintexts into an (encrypted) IBLT. The encrypted IBLT would then constitute the compression of the vector.

This approach encounters two problems: First, the insertion operation of an IBLT requires hashing the value to choose the cells to insert it in, which we cannot do because we do not have access to the plaintext. Second, since we do not *know* which of the ciphertexts correspond to a non-zero it is unclear how to only insert those.

The first problem can be solved using Remark 1 by actually storing pairs (d, m_d). Since the index d is both publically known and unique, we can rely on only hashing d to derive the positions to insert the values.

The second problem is a bit trickier to solve. To build some intuition, we can first consider an easier compression problem where in addition to $\boldsymbol{c}$ we are given an additional vector of ciphertexts $\boldsymbol{h}$ containing "zero hints". I.e., h_d decrypts to 0 if c_i decrypts to 0 and h_d decrypts to 1, if c_d decrypts to anything else.

The IBLT then gets initialized as three matrices, a count matrix $\boldsymbol{C}$ and two valueSum matrices $\boldsymbol{M}$ and $\boldsymbol{D}$ with each cell containing an encryption of zero. To insert the content of ciphertext c_d into the IBLT, we can then for $i \in [\gamma]$ compute $j := h_i(d)$ and insert the value by setting $\boldsymbol{C}[i,j] := \boldsymbol{C} \boxplus h_d$, $\boldsymbol{M}[i,j] := \boldsymbol{M} \boxplus c_d$, and $\boldsymbol{D}[i,j] := \boldsymbol{D} \boxplus (d \boxdot h_d)$.

Note that for any ciphertext corresponding to zero, this results in zero being added to all entries, which is equivalent to not inserting the value at all. Decompressing then involves decrypting all three matrices and using the List algorithm to extract pairs (d, m_d) giving us a sparse representation of the plaintext vector corresponding to $\boldsymbol{c}$. By the correctness guarantee of an IBLT, this works as long as not too many ciphertexts decrypt to a non-zero value.

However, actually getting such "zero hint" ciphertexts may not be feasible in all scenarios, especially if the encryption scheme is *only* additively homomorphic. This means we need to somehow simulate having a count matrix without these zero hints.

The trick that we use is to choose a vector of random values $\boldsymbol{k}$ that we will use to "recognize" cells that only contain a single message. We will still initialize two matrices $\boldsymbol{M}$ and $\boldsymbol{K}$ but inserting into the IBLT is now done by setting $\boldsymbol{M}[i,j] := \boldsymbol{M} \boxplus c_d$, and $\boldsymbol{K}[i,j] := \boldsymbol{K} \boxplus (k_d \boxdot c_d)$. Note now, that after decryption, for any cell (i,j) that only contains a single value m_d, we have that $\boldsymbol{M}[i,j] = m_d$ and $\boldsymbol{K}[i,j] = k_d \cdot m_d$. By checking if $\boldsymbol{K}[i,j]/\boldsymbol{M}[i,j]$ corresponds to one of the values in $\boldsymbol{k}$, we can thus recognize which cells contain only a single value and which index it corresponds to, allowing us to peel the message from the IBLT. In section we prove in a helpful lemma in Section 5.2 we prove that we can bound the probability that this recognition procedure produces false positives.

There still remains the problem that simply using a random vector $\boldsymbol{k}$ and storing it, which would require $\mathcal{O}(n)$ storage and $\mathcal{O}(n)$ computation to recognize the entries. To solve this issue we introduce the concept of wunderbar pseudorandom vectors in Section 5.1, which allows us to store a compact $\mathcal{O}(\lambda)$ representation of a pseudorandom vector $\boldsymbol{k}$ and recognition of vector entries in time $\mathcal{O}(\mathsf{polylog}(n))$.

5.1 Wunderbar Pseudorandom Vectors

The concept of a pseudorandom vector is conceptually similar to that of pseudorandom sets introduced in [CK20], except that we do not require puncturability. The idea is that it allows us to sample a short description of a long vector, which is indistinguishable from a random vector with unique entries. Importantly, we require that there exists an efficient algorithm that can recover the position of a given entry in the vector in time independent of the vector length. Naively one can always find the position in linear time in the vector length. This is, however, not good enough for our application, which is why we require the pseudorandom vector to be "wunderbar". In particular, we want the description length of the vector to be in $\mathcal{O}(\lambda)$ and getting individual entries as well as index recovery should be possible in $\mathcal{O}(\mathsf{polylog}(n))$.

Definition 7. *A pseudorandom vector with index recovery for an efficiently sampleable universe $K = K(\lambda)$ consists of a triple of ppt algorithms* $(\mathsf{Sample}, \mathsf{Entry}, \mathsf{Index})$ *such that*

$\mathsf{Sample}(1^\lambda, n)$**:** *The sampling algorithm takes as input the security parameter λ and the vector length n in unary and outputs the description of a pseudorandom vector s.*

$\mathsf{Entry}(s, i)$**:** *The deterministic retrieving algorithm takes as input a description s and an index $i \in [n]$ and outputs a value $k_i \in K$.*

$\mathsf{Index}(s, k)$**:** *The deterministic index recovery algorithm takes as input a description s and a value k and outputs either an index $i \in [n]$ or $\perp$.*

A pseudorandom vector with index recovery is correct, if for all vector lengths $n = \mathsf{poly}(\lambda)$ and all seeds $s \leftarrow \mathsf{Gen}(1^\lambda, 1^n)$ it holds that:

1. *For all indices $i \in [n]$ it holds that $\mathsf{Index}(s, \mathsf{Entry}(s, i)) = i$.*
2. *For all all $k^* \notin \{\mathsf{Entry}(s, i) \mid i \in [n]\}$ it holds that $\mathsf{Index}(s, k^*) = \perp$.*

The pseudorandom vector is wunderbar *if the description of a vector has length $\mathcal{O}(\lambda)$ and the runtime of* Entry *and* Index *is $\mathcal{O}(\mathsf{polylog}(n))$. A pseudorandom vector is secure, if for all $n = \mathsf{poly}(\lambda)$ and all ppt algorithms $\mathcal{A}$*

$$\left| \Pr\left[\begin{aligned} & s \leftarrow \mathsf{Sample}(1^\lambda, 1^n), \\ & \boldsymbol{k} := \begin{pmatrix} \mathsf{Entry}(s,1) \\ \vdots \\ \mathsf{Entry}(s,n) \end{pmatrix} \end{aligned} : \mathcal{A}(\boldsymbol{k}) \right] - \Pr[\boldsymbol{k} \leftarrow ✄(K^n) : \mathcal{A}(\boldsymbol{k})] \right| \leq \mathsf{negl}(\lambda)$$

Remark 3. For ease of notation we define two algorithms $\mathsf{DummySample}$ and $\mathsf{DummyIndex}$ that represent a dummy version of a pseudorandom vector with index recovery. I.e., $\mathsf{DummySample}(1^\lambda, n)$ simply samples $\boldsymbol{k} \leftarrow ✄(K^n)$ and $\mathsf{DummyIndex}(\boldsymbol{k}, k)$ performs an exhaustive search and returns i iff $k_i = k$ and $\perp$ if none of them match. Using this notation, the above security definition can be rewritten as

$$\left| \Pr\left[\begin{aligned} & s \leftarrow \mathsf{Sample}(1^\lambda, 1^n), \\ & \boldsymbol{k} := \begin{pmatrix} \mathsf{Entry}(s,1) \\ \vdots \\ \mathsf{Entry}(s,n) \end{pmatrix} \end{aligned} : \mathcal{A}(\boldsymbol{k}) \right] - \Pr[\boldsymbol{k} \leftarrow \mathsf{DummySample}(1^\lambda, n) : \mathcal{A}(\boldsymbol{k})] \right| \leq \mathsf{negl}(\lambda)$$

Wunderbar Pseudorandom Vectors from Pseudorandom Permuations Let $\mathbb{F}_{p^m}$ be a field such that $m \cdot \lfloor \log p \rfloor \geq \lambda$. We construct wunderbar pseudorandom vectors over a subset $K \subseteq \mathbb{F}_{p^m}$ from an arbitrary family of pseudorandom permutations over $\mathbb{F}_2^\lambda$. To do so we need an efficiently computable and efficiently invertible injective function $\mathsf{binToField}$ mapping from $\mathbb{F}_2^\lambda$ to $\mathbb{F}_{p^m}$. The exact function is irrelevant, but for concreteness, we specify it in the following. Let $\{0,1\} : [q] \to \mathbb{F}_2^{\lceil \log q \rceil}$ denote the function that maps an integer to its canonical binary representation and let $\mathsf{proj} : \mathbb{F}_2^{\lceil \log q \rceil} \to [q]$ be its inverse. Then we specify

$$\mathsf{binToField} : \mathbb{F}_2^\lambda \to \mathbb{F}_{p^m} \quad \mathsf{binToField}((b_1, \ldots, b_\lambda)) := \sum_{i=0}^{m-1} c_i x^i$$

where

$$c_i := \sum_{j=1}^{\min\{\lceil \frac{\lambda}{m} \rceil, \lambda - i \lceil \frac{\lambda}{m} \rceil\}} 2^{j-1} b_{i\lceil \frac{\lambda}{m} \rceil + j}.$$

We further specify the inverse function as

$$\mathsf{fieldToBin} : \mathbb{F}_{p^m} \to \mathbb{F}_2^\lambda \cup \{\perp\}$$
$$\mathsf{fieldToBin}(\sum_{i=0}^{m-1} c_i x^i) := \begin{cases} \perp & \text{if } \exists c_i.\ c_i \geq 2^{\min\{\lceil \frac{\lambda}{m} \rceil, \lambda - i\lceil \frac{\lambda}{m} \rceil\}} \\ (b_1, \ldots, b_\lambda) & \text{otherwise} \end{cases}$$

where

$$b_i := \mathsf{bin}\big(c_{\lfloor i/\lceil \lambda/m \rceil \rfloor}\big)_{i - \lfloor i/\lceil \lambda/m \rceil \rfloor \cdot \lceil \lambda/m \rceil}.$$

```
Sample($1^\lambda, 1^n$)
$s \leftarrow \mathbb{F}_2^\lambda$
return $(s, n)$
```

```
Entry($(s, n), i$)
$\boldsymbol{b} := \mathsf{bin}(i)$
$\boldsymbol{b}' := \mathsf{PRP}(s, \boldsymbol{b})$
return $\mathsf{binToField}(\boldsymbol{b}')$
```

```
Index($(s, n), k$)
$\boldsymbol{b}' := \mathsf{fieldToBin}(k)$
if $\boldsymbol{b}' \neq \perp$
  $\boldsymbol{b} := \mathsf{PRP}^{-1}(s, \boldsymbol{b}')$
  if $\mathsf{proj}(\boldsymbol{b}) \in [n]$
    return $\mathsf{proj}(\boldsymbol{b})$
return $\perp$
```

Fig. 3. A wunderbar pseudorandom vector for $K \subseteq \mathbb{F}_{p^m}$ constructed from a family of pseudorandom permuations over $\mathbb{F}_2^\lambda$.

Theorem 5. *Let* PRP *be a secure family of pseudorandom permuations over some* $\mathbb{F}_2^\lambda$*. Then* ($\mathsf{Sample}, \mathsf{Entry}, \mathsf{Index}$) *as described in Figure 3 is a secure wunderbar pseudorandom vector with index recovery for universe* $K = \{\mathsf{binToField}(\boldsymbol{b}) \mid \boldsymbol{b} \in \mathbb{F}_2^\lambda\}$.

Proof. We need to establish that the construction is correct, wunderbar, and secure. It is simple to see that the construction is correct:

$$\begin{aligned}
&\mathsf{Index}((s,n), \mathsf{Entry}((s,n), i)) \\
=&\mathsf{Index}((s,n), \mathsf{binToField}(\mathsf{PRP}(s, \mathsf{bin}(i)))) && \text{(Def. of Entry)} \\
=&\mathsf{proj}(\mathsf{PRP}^{-1}(s, \mathsf{fieldToBin}(\mathsf{binToField}(\mathsf{PRP}(s, \mathsf{bin}(i)))))) && \text{(Def. of Index)} \\
=&\mathsf{proj}(\mathsf{PRP}^{-1}(s, \mathsf{PRP}(s, \mathsf{bin}(i))))
\end{aligned}$$

$$\begin{aligned}&=\mathsf{proj}(\mathsf{bin}(i))\\&=i.\end{aligned}$$

Similarly, it is easy to see that the construction is wunderbar: the description consists of $s \in \mathbb{F}_2^\lambda$ and $n = \mathsf{poly}(\lambda) \leq \lambda^c$ for some constant c. Therefore, it has size at most $\lambda + c \cdot \log \lambda \in \mathcal{O}(\lambda)$. The runtime of Entry is in fact independent of n and thus trivially in $\mathcal{O}(\mathsf{polylog}(n))$ and the only computation in Index that depends on n, is the membership check $\mathsf{proj}(\boldsymbol{b}) \in [n]$ which can be performed in time $\mathcal{O}(\log n) \subset \mathcal{O}(\mathsf{polylog}(n))$.

It remains to show that the construction is secure. Let $n = n(\lambda) = \mathsf{poly}(\lambda)$ and let $\mathcal{A}$ be an arbitrary PPT algorithm, such that We construct an adversary $\mathcal{B}$ against the pseudorandomness of as follows. $\mathcal{B}$ takes as input the security parameter λ and is given access to an oracle. For each $i \in [n]$, query $\mathsf{bin}(i)$ to the oracle, receiving back $\boldsymbol{b}_i'$ and compute $k_i := \mathsf{binToField}(\boldsymbol{b}_i')$. Invoke $\mathcal{A}(\boldsymbol{k})$ and output whatever $\mathcal{A}$ outputs. Clearly, $\mathcal{B}$ is also PPT, needing a runtime overhead of just n oracle queries over simply running $\mathcal{A}$. We now consider two cases: if, on the one hand, the oracle is $\mathsf{PRP}(s, \cdot)$, then for all $i \in [n]$ $k_i = \mathsf{binToField}(\mathsf{PRP}(s, \mathsf{bin}(i))) = \mathsf{Entry}((s, n), i)$. I.e., we have

$$\begin{aligned}&\Pr[s \leftarrow \mathbb{F}_2^\lambda : \mathcal{B}^{\mathsf{PRP}(s,\cdot)} = 1]\\=&\Pr[s \leftarrow \mathsf{Sample}(1^\lambda, 1^n), \boldsymbol{k} := (\mathsf{Entry}(s,1), \ldots, \mathsf{Entry}(s,n))^\intercal) : \mathcal{A}(\boldsymbol{k})].\end{aligned} \tag{1}$$

If, on the other hand, the oracle is a truly random permutation g, then for all $i \in [n]$ it holds that $k_i = \mathsf{binToField}(g(\mathsf{bin}(i)))$ and therefore

$$\begin{aligned}&\Pr[g \leftarrow \mathit{\Pi}(\mathbb{F}_2^\lambda) : \mathcal{B}^{g(\cdot)} = 1]\\=&\Pr[g \leftarrow \mathit{\Pi}(\mathbb{F}_2^\lambda); \forall i \in [n].\, k_i = \mathsf{binToField}(g(\mathsf{bin}(i))) : \mathcal{A}(\boldsymbol{k})] &(2)\\=&\Pr[(\boldsymbol{b}_1', \ldots, \boldsymbol{b}_n') \leftarrow \text{✂}\big((\mathbb{F}_2^\lambda)^n\big); \boldsymbol{k} = (\mathsf{binToField}(\boldsymbol{b}_i'))_{i \in [n]} : \mathcal{A}(\boldsymbol{k})] &(3)\\=&\Pr[\boldsymbol{k} \leftarrow \text{✂}(K^n) : \mathcal{A}(\boldsymbol{k})]. &(4)\end{aligned}$$

Here, Equation 3 holds because g is a uniformly chosen random permutation and therefore the values $g(\mathsf{bin}(i))$ are uniformly distributed conditioned on not being duplicates and Equation 4 holds because $\mathsf{binToField}$ is an injective function into K.

Combining Equation 1 and Equation 4 we get

$$\begin{aligned}&\left|\Pr\left[\begin{array}{l}s \leftarrow \mathsf{Sample}(1^\lambda, 1^n),\\ \boldsymbol{k} := \begin{pmatrix}\mathsf{Entry}(s,1)\\ \vdots \\ \mathsf{Entry}(s,n)\end{pmatrix}\end{array} : \mathcal{A}(\boldsymbol{k})\right] - \Pr[\boldsymbol{k} \leftarrow \text{✂}(K^n) : \mathcal{A}(\boldsymbol{k})]\right|\\=&\left|\Pr[s \leftarrow \mathbb{F}_2^\lambda : \mathcal{B}^{\mathsf{PRP}(s,\cdot)} = 1] - \Pr[g \leftarrow \{f : \mathbb{F}_2^\lambda \to \mathbb{F}_2^\lambda\} : \mathcal{B}^{g(\cdot)} = 1]\right|\\\leq&\mathsf{negl}(\lambda)\end{aligned}$$

where the last inequality follows from the fact that PRP is pseudorandom. □

5.2 A Helpful Lemma

We prove a helpful lemma which allows to bound the probability of false positives when attempting to detect cells with only a single entry in the IBLT. Recall, that we have two matrices $\boldsymbol{M}$ and $\boldsymbol{K}$, where the cells of $\boldsymbol{M}$ contain sums of messages m_d and the cells of $\boldsymbol{K}$ contain sums of $k_d \cdot m_d$ for a random vector $\boldsymbol{k}$. We check for cells containing only a single message, i.e. cells that can be peeled, by checking whether $\boldsymbol{K}[i,j]/\boldsymbol{M}[i,j]$ corresponds to one of the values in $\boldsymbol{k}$. A false positive could occur, if for some set of at least two non-zero messages corresponding to indices $I \subseteq [n]$ it happens to hold that

$$k_j = \frac{\sum_{i \in I} k_i m_i}{\sum_{i \in I} m_i}$$

for some $j \in [n]$. The lemma states that we can bound the probability of this occuring by choosing the entries of $\boldsymbol{k}$ from a large enough space.

Lemma 6. *Let $K \subseteq \mathbb{F}_q$, $(m_1, \dots, m_n) \in \mathbb{F}_q^n$ and $I \subseteq [n]$ be arbitrary such that $\sum_{i \in I} m_i \neq 0$ and there exist $i, i' \in I$ with $0 \notin \{m_i, m_{i'}\}$. It holds that*

$$\Pr\left[\boldsymbol{k} \leftarrow K^n : \exists j \in [n].\, k_j = \frac{\sum_{i \in I} k_i m_i}{\sum_{i \in I} m_i}\right] \leq \frac{n}{|K|}$$

Proof. Using a union bound we have

$$\Pr\left[\boldsymbol{k} \leftarrow K^n : \exists j \in [n].\, k_j = \frac{\sum_{i \in I} k_i m_i}{\sum_{i \in I} m_i}\right] \leq \sum_{j \in [n]} \Pr\left[\boldsymbol{k} \leftarrow K^n : k_j = \frac{\sum_{i \in I} k_i m_i}{\sum_{i \in I} m_i}\right]. \tag{5}$$

It thus remains to bound the above probability for individual j. Let $j \in [n]$ be arbitrary but fixed and let $\xi = 1$ if $j \in I$ and $\xi = 0$ otherwise. It then holds that

$$\begin{aligned} &\Pr\left[\boldsymbol{k} \leftarrow K^n : k_j = \frac{\sum_{i \in I} k_i m_i}{\sum_{i \in I} m_i}\right] \\ =&\Pr\left[\boldsymbol{k} \leftarrow K^n : k_j \cdot \sum_{i \in I} m_i = \sum_{i \in I} k_i m_i\right] \\ =&\Pr\left[\boldsymbol{k} \leftarrow K^n : k_j \cdot \big(\sum_{i \in I} m_i - \xi m_j\big) = \sum_{i \in I} k_i m_i - k_j \xi m_j\right] \\ =&\Pr\left[\boldsymbol{k} \leftarrow K^n : k_j \cdot \sum_{i \in I \setminus \{j\}} m_i = \sum_{i \in I \setminus \{j\}} k_i m_i\right] \end{aligned}$$

We now consider two cases. If $\sum_{i \in I \setminus \{j\}} = 0$, let $j' \in I \setminus \{j\}$ be an index, such that $m_{j'} \neq 0$. Note that such an index always exists by the condition on I. We then have

$$\begin{aligned} &\Pr\left[\boldsymbol{k} \leftarrow K^n : k_j \cdot \overbrace{\sum_{i \in I \setminus \{j\}} m_i}^{=0} = \sum_{i \in I \setminus \{j\}} k_i m_i\right] \\ =&\Pr\left[\boldsymbol{k} \leftarrow K^n : 0 = \sum_{i \in I \setminus \{j\}} k_i m_i\right] \\ =&\Pr\left[\boldsymbol{k} \leftarrow K^n : k_{j'} = \frac{\sum_{i \in I \setminus \{j, j'\}} k_i m_i}{-m_{j'}}\right] \end{aligned}$$

where $(-m_{j'})^{-1}$ is always defined by the condition that $m_{j'} \neq 0$. Since the right hand side of the equality is independent of $k_{j'}$, the probability that the equality holds is at most $1/|K|$ for any choice of k_i, $i \neq j'$. Thus, in this case

$$\Pr\left[\boldsymbol{k} \leftarrow K^n : k_j = \frac{\sum_{i \in I} k_i m_i}{\sum_{i \in I} m_i}\right] \leq \frac{1}{|K|}. \tag{6}$$

In the other case, i.e., if $\sum_{i \in I \setminus \{j\}} \neq 0$, $(\sum_{i \in I \setminus \{j\}})^{-1}$ is well defined and we have

$$\Pr\left[\boldsymbol{k} \leftarrow K^n : k_j \cdot \sum_{i \in I \setminus \{j\}} m_i = \sum_{i \in I \setminus \{j\}} k_i m_i\right] = \Pr\left[\boldsymbol{k} \leftarrow K^n : k_j = \frac{\sum_{i \in I \setminus \{j\}} k_i m_i}{\sum_{i \in I \setminus \{j\}} m_i}\right].$$

Here again, the right hand side of the equality is independent of k_j. Thus, the probability that the equality holds is at most $1/|K|$ for any choice of k_i, $i \neq j$ and also in this case it holds that

$$\Pr\left[\boldsymbol{k} \leftarrow K^n : k_j = \frac{\sum_{i \in I} k_i m_i}{\sum_{i \in I} m_i}\right] \leq \frac{1}{|K|} \tag{7}$$

Finally, combining Equation 5 with Equation 6 and Equation 7, we get

$$\Pr\left[\boldsymbol{k} \leftarrow K^n : \exists j \in [n].\, k_j = \frac{\sum_{i\in I} k_i m_i}{\sum_{i \in I} m_i}\right] \leq \frac{n}{|K|} \qquad \square$$

By observing that the statistical distance betweem K^n and $✂(K^n))$ is at most $n^2/|K|$ due to the birthday bound, we obtain the following corollary.

Corollary 7. *Let $K \subseteq \mathbb{F}_q$, $(m_1, \ldots, m_n) \in \mathbb{F}_q^n$ and $I \subseteq [n]$ be arbitrary such that $\sum_{i\in I} m_i \neq 0$ and there exist $i, i' \in I$ with $0 \notin \{m_i, m_{i'}\}$. It holds that*

$$\Pr\left[\boldsymbol{k} \leftarrow ✂(K^n) : \exists j \in [n].\, k_j = \frac{\sum_{i\in I} k_i m_i}{\sum_{i \in I} m_i}\right] \leq \frac{n^2+n}{|K|}$$

5.3 Construction of Ciphertext-Compression from IBLTs

Populating the IBLT involves homomorphically evaluating an inner product between the encrypted vector and a plain vector. Therefore, the compression scheme can only work for ciphertext vectors that allow the evaluation of inner product functions defined in the following.

Definition 8 (Inner Product Functions). *The class of inner product functions is the set of functions $\mathcal{Z}_{\mathsf{ip}} = \{f_{\boldsymbol{a}} \mid \boldsymbol{a} \in \mathbb{F}_q^n\}$ with*

$$f_{\boldsymbol{a}} : \mathbb{F}_q^n \to \mathbb{F}_q, \quad f_{\boldsymbol{a}}(\boldsymbol{x}) := \langle \boldsymbol{a}, \boldsymbol{x} \rangle.$$

The family of ciphertext vectors the construction is applicable to is then exactly those ciphertext vectors with low hamming weight and allow the evaluation of inner product functions. We define this family as follows.

Definition 9 ($\mathcal{Z}_{\mathsf{ip}}$-Valid Low Hamming Weight Ciphertext Vectors). *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Enc}, \mathsf{Dec})$ be a homomorphic public key encryption scheme. For any $\mathsf{pk} \in \mathcal{P}$, let*

$$\mathcal{F}^{\mathsf{ip}}_{t,\mathsf{pk}} := \left\{\boldsymbol{c} \in \mathsf{vld}(\mathcal{Z}_{\mathsf{ip}}, \mathsf{pk}) \mid \mathsf{hw}(\mathsf{Dec}(\mathsf{Gen}^{-1}(\mathsf{pk}), \boldsymbol{c})) < t\right\}.$$

We then define the family of $\mathcal{Z}_{\mathsf{ip}}$-valid ciphertext vectors with low hamming weight as $\mathcal{F}^{\mathsf{ip}}_t := \{\mathcal{F}_{t,\mathsf{pk}} \mid \mathsf{pk} \in \mathcal{P}\}$.

Theorem 8. *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be an additively homomorphic encryption scheme with message space $\mathcal{M} = \mathbb{F}_q$ for some prime power q with ciphertext size $\xi = \xi(\lambda)$. Let $\lambda, \kappa, t, n \in \mathbb{N}$ be integers and let $\gamma := \frac{\kappa}{\log t}$. Let PRF be a family of pseudorandom functions $\mathsf{PRF} : [\gamma] \times [2^\lambda] \to [2t]$ and let $(\mathsf{Sample}, \mathsf{Entry}, \mathsf{Index})$ be a wunderbar pseudorandom vector with index recovery for a universe $K = K(\lambda) \subseteq \mathbb{F}_q$ with $|K| \geq 2^\kappa (2t\gamma)(n^3 + n^2)$. Then $(\mathsf{Compress}, \mathsf{Decompress})$ from Figure 4 is a $(\mathcal{O}(\lambda) + 4t\gamma\xi)/(n\xi)$-compressing $(1 - \mathcal{O}(2^{-\kappa}) - \mathsf{negl}(\lambda))$-correct ciphertext compression scheme for family $\mathcal{F}^{\mathsf{ip}}_t$.*

Proof. The output of the compression algorithm consists of a s_1, s_2 and $4t\gamma$ ciphertexts. Since the pseudorandom vector is wunderbar, it holds that $|s_1| = \mathcal{O}(\lambda)$ and s_2 is chosen as a λ-bit string. Therefore, it is easy to see that the scheme is $(\mathcal{O}(\lambda) + 4t\gamma\xi)/(n\xi)$-compressing. It remains to prove that it is correct. To do so we define a series of six hybrid schemes in Figures 5 through 9.

Claim 9. *For any $\mathcal{Z}_{\mathsf{ip}}$-valid vector of ciphertexts it holds that*

$$\begin{aligned} &\Pr[\mathsf{Decompress}(\mathsf{sk}, \mathsf{Compress}(\mathsf{pk}, \boldsymbol{c})) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))] \\ = &\Pr[\mathsf{Decompress}_1(\mathsf{Compress}_1(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))] \end{aligned}$$

```
Compress(pk, $\boldsymbol{c}$)
$s_1 \leftarrow \mathsf{Sample}(1^\lambda, n)$
$s_2 \leftarrow \{0,1\}^\lambda$
$\boldsymbol{M} := \mathsf{Enc}(\mathsf{pk}, 0)^{\gamma\times 2t}$
$\boldsymbol{K} := \mathsf{Enc}(\mathsf{pk}, 0)^{\gamma\times 2t}$
foreach $(i,d) \in [\gamma]\times[n]$ do
  $j := \mathsf{PRF}(s_2, (i,d))$
  $\boldsymbol{M}[i,j] := \boldsymbol{M}[i,j] \boxplus c_d$
  $k := \mathsf{Entry}(s_1, d)$
  $\boldsymbol{K}[i,j] := \boldsymbol{K}[i,j] \boxplus (k \boxdot c_d)$
return $(s_1, s_2, \boldsymbol{M}, \boldsymbol{K})$
```

```
Decompress(sk, $(s_1, s_2, \boldsymbol{M}, \boldsymbol{K})$)
$S := \emptyset$
$\boldsymbol{M}' := \mathsf{Dec}(\mathsf{sk}, \boldsymbol{M})$
$\boldsymbol{K}' := \mathsf{Dec}(\mathsf{sk}, \boldsymbol{K})$
$\boldsymbol{D}' := \mathsf{Initialize}()$
while $\exists\,(i^*, j^*) \in [\gamma]\times[2t].\ \boldsymbol{D}'[i^*,j^*] \neq \bot$ do
  $(d,k,m) := (\boldsymbol{D}'[i^*,j^*], \boldsymbol{K}'[i^*,j^*], \boldsymbol{M}'[i^*,j^*])$
  $S := S \cup \{(d,m)\}$
  $\mathsf{Update}(d,k,m)$
return $S$
```

```
Initialize()
$\boldsymbol{D}' := \bot^{\gamma\times 2t}$
foreach $(i,j) \in [\gamma]\times[2t]$ do
  if $\boldsymbol{M}'[i,j] \neq 0$
    $\boldsymbol{D}'[i,j] := \mathsf{Index}\left(s_1, \frac{\boldsymbol{K}'[i,j]}{\boldsymbol{M}'[i,j]}\right)$
return $\boldsymbol{D}'$
```

```
Update($d, k, m$)
foreach $i \in [\gamma]$ do
  $j := \mathsf{PRF}(s_2, (i,d))$
  $\boldsymbol{M}'[i,j] := \boldsymbol{M}'[i,j] - m$
  $\boldsymbol{K}'[i,j] := \boldsymbol{K}'[i,j] - k$
  if $\boldsymbol{M}'[i,j] \neq 0$
    $\boldsymbol{D}'[i,j] := \mathsf{Index}\left(s_1, \frac{\boldsymbol{K}'[i,j]}{\boldsymbol{M}'[i,j]}\right)$
  else
    $\boldsymbol{D}'[i,j] := \bot$
```

Fig. 4. A ciphertext compression scheme based on invertible bloom lookup tables and wunder pseudorandom vectors.

```
Compress$_1$($\boldsymbol{m}$)
$s_1 \leftarrow \mathsf{Sample}(1^\lambda, n)$
$s_2 \leftarrow \{0,1\}^\lambda$
$\boldsymbol{M} := 0^{\gamma\times 2t}$
$\boldsymbol{K} := 0^{\gamma\times 2t}$
foreach $(i,d) \in [\gamma]\times[n]$ do
  $j := \mathsf{PRF}(s_2, (i,d))$
  $\boldsymbol{M}[i,j] := \boldsymbol{M}[i,j] + m_d$
  $k := \mathsf{Entry}(s_1, d)$
  $\boldsymbol{K}[i,j] := \boldsymbol{K}[i,j] + (k \cdot m_d)$
return $(s_1, s_2, \boldsymbol{M}, \boldsymbol{K})$
```

```
Decompress$_1$($(s_1, s_2, \boldsymbol{M}, \boldsymbol{K})$)
$S := \emptyset$
$\boldsymbol{M}' := \boldsymbol{M}$
$\boldsymbol{K}' := \boldsymbol{K}$
$\boldsymbol{D}' := \mathsf{Initialize}()$
while $\exists\,(i^*, j^*) \in [\gamma]\times[2t].\ \boldsymbol{D}'[i^*,j^*] \neq \bot$ do
  $(d,k,m) := (\boldsymbol{D}'[i^*,j^*], \boldsymbol{K}'[i^*,j^*], \boldsymbol{M}'[i^*,j^*])$
  $S := S \cup \{(d,m)\}$
  $\mathsf{Update}(d,k,m)$
return $S$
```

Fig. 5. The first hybrid scheme works exactly as the actual ciphertext compression scheme, except that it operates on *plaintext* messages instead of encrypted messages. I.e., ciphertexts are now decrypted before compression instead of between compression and decompression.

Proof. Following Definition 8 we denote by $f_{\boldsymbol{a}}$ the inner product function $f_{\boldsymbol{a}} : \mathbb{F}_q^n \to \mathbb{F}_q, \quad f_{\boldsymbol{a}}(\boldsymbol{x}) := \langle \boldsymbol{a}, \boldsymbol{x} \rangle$. Further, we denote

$$\boldsymbol{v}_{i,j} := \begin{pmatrix} \delta_{j,\mathsf{PRF}(s_2,(i,1))} \\ \vdots \\ \delta_{j,\mathsf{PRF}(s_2,(i,n))} \end{pmatrix} \quad \boldsymbol{w}_{i,j} := \begin{pmatrix} \mathsf{Entry}(s_1,1) \cdot \delta_{j,\mathsf{PRF}(s_2,(i,1))} \\ \vdots \\ \mathsf{Entry}(s_1,n) \cdot \delta_{j,\mathsf{PRF}(s_2,(i,n))} \end{pmatrix}$$

Now, let $\boldsymbol{M}_0, \boldsymbol{K}_0, \boldsymbol{M}'_0, \boldsymbol{K}'_0$ and $\boldsymbol{M}_1, \boldsymbol{K}_1, \boldsymbol{M}'_1, \boldsymbol{K}'_1$ denote the relevant matrices in the actual scheme and hybrid 1 respectively. We note, that since $\boldsymbol{c}$ is $\mathcal{Z}_{\mathsf{ip}}$-valid, it holds for all $(i,j) \in [\gamma] \times [2t]$ that

$$\begin{aligned} \boldsymbol{M}'_0[i,j] =& \mathsf{Dec}(\mathsf{sk}, \boldsymbol{M}_0[i,j]) \\ =& \mathsf{Dec}(\mathsf{sk}, \boxplus_{d \in \{[n] \mid \mathsf{PRF}(s_2,(i,d))=j\}} c_d) \\ =& \mathsf{Dec}(\mathsf{Eval}(\mathsf{pk}, f_{\boldsymbol{v}_{i,j}}, \boldsymbol{c})) \\ =& f_{\boldsymbol{v}_{i,j}}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c})) && \text{(Definition 3)} \\ =& \sum_{d \in \{[n] \mid \mathsf{PRF}(s_2,(i,d))=j\}} \mathsf{Dec}(\mathsf{sk}, c_d) = \boldsymbol{M}'_1[i,j] \end{aligned}$$

as well as

$$\begin{aligned} \boldsymbol{K}'_0[i,j] =& \mathsf{Dec}(\mathsf{sk}, \boldsymbol{K}_0[i,j]) \\ =& \mathsf{Dec}(\mathsf{sk}, \boxplus_{d \in \{[n] \mid \mathsf{PRF}(s_2,(i,d))=j\}} c_d \cdot \mathsf{Entry}(s_1, d)) \\ =& \mathsf{Dec}(\mathsf{Eval}(\mathsf{pk}, f_{\boldsymbol{w}_{i,j}}, \boldsymbol{c})) \\ =& f_{\boldsymbol{w}_{i,j}}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c})) && \text{(Definition 3)} \\ =& \sum_{d \in \{[n] \mid \mathsf{PRF}(s_2,(i,d))=j\}} \mathsf{Dec}(\mathsf{sk}, c_d) \cdot \mathsf{Entry}(s_1,d)) = \boldsymbol{K}'_1[i,j] \end{aligned}$$

Since the computation on $\boldsymbol{M'}, \boldsymbol{K'}$ is otherwise identical between the two hybrids, the claim immediately follows. □

Claim 10. *If* $(\mathsf{Sample}, \mathsf{Entry}, \mathsf{Index})$ *is a secure pseudorandom vector, it holds for any key pair* $(\mathsf{sk}, \mathsf{pk})$ *and any vector* $\boldsymbol{c}$ *that*

$$\left| \begin{array}{l} \Pr[\mathsf{Decompress}_1(\mathsf{Compress}_1(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))] \\ - \Pr[\mathsf{Decompress}_2(\mathsf{Compress}_2(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

Proof. We construct an attacker $\mathcal{A}$ against security of the pseudorandom vector as follows. On input $\boldsymbol{k}$, $\mathcal{A}$ executes $\mathsf{Decompress}_2(\mathsf{Compress}_2(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c})))$, except that it uses its input $\boldsymbol{k}$ instead of sampling a fresh one. If $\boldsymbol{k}$ was chosen using $\boldsymbol{k} \leftarrow \mathsf{DummySample}(1^\lambda, n)$, this is identical to a regular execution of $\mathsf{Decompress}_2(\mathsf{Compress}_2(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c})))$. If on the other hand $\boldsymbol{k}$ was chosen by sampling $s_2 \leftarrow \mathsf{Sample}(1^\lambda, n)$ and setting $\boldsymbol{k} := (\mathsf{Entry}(s,1), \ldots, \mathsf{Entry}(s,n))^\intercal$, this is identical to a regular execution of $\mathsf{Decompress}_1(\mathsf{Compress}_1($ $\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c})))$. Therefore, by the security of the pseudorandom vector

$$\begin{aligned} \mathsf{negl}(\lambda) \geq& \left| \begin{array}{l} \Pr\left[s \leftarrow \mathsf{Sample}(1^\lambda, n), \boldsymbol{k} := \begin{pmatrix} (\mathsf{Entry}(s,1) \\ \vdots \\ \mathsf{Entry}(s,n) \end{pmatrix} : \mathcal{A}(\boldsymbol{k}) \right] \\ - \Pr[\boldsymbol{k} \leftarrow \mathsf{DummySample}(1^\lambda, n) : \mathcal{A}(\boldsymbol{k})] \end{array} \right| \\ =& \left| \begin{array}{l} \Pr[\mathsf{Decompress}_1(\mathsf{sk}, \mathsf{Compress}_1(\mathsf{pk}, \boldsymbol{c})) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))] \\ - \Pr[\mathsf{Decompress}_2(\mathsf{Compress}_2(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))] \end{array} \right| \end{aligned}$$

□

```
$\mathsf{Compress}_2(\boldsymbol{m})$
$\boldsymbol{k} \leftarrow \mathsf{DummySample}(1^\lambda, n)$
$s_2 \leftarrow \{0,1\}^\lambda$
$\boldsymbol{M} := 0^{\gamma \times 2t}$
$\boldsymbol{K} := 0^{\gamma \times 2t}$
foreach $(i, d) \in [\gamma] \times [n]$ do
  $j := \mathsf{PRF}(s_2, (i, d))$
  $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] + m_d$
  $k := k_d$
  $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] + (k \cdot m_d)$
return $(\boldsymbol{k}, s_2, \boldsymbol{M}, \boldsymbol{K})$
```

```
$\mathsf{Decompress}_2((\boldsymbol{k}, s_2, \boldsymbol{M}, \boldsymbol{K}))$
$S := \emptyset$
$\boldsymbol{M}' := \boldsymbol{M}$
$\boldsymbol{K}' := \boldsymbol{K}$
$\boldsymbol{D}' := \mathsf{Initialize}_2()$
while $\exists\, (i^*, j^*) \in [\gamma] \times [2t].\ \boldsymbol{D}'[i^*, j^*] \neq \perp$ do
  $(d, k, m) := (\boldsymbol{D}'[i^*, j^*], \boldsymbol{K}'[i^*, j^*], \boldsymbol{M}'[i^*, j^*])$
  $S := S \cup \{(d, m)\}$
  $\mathsf{Update}_2(d, k, m)$
return $S$
```

```
$\mathsf{Initialize}_2()$
$\boldsymbol{D}' := \perp^{\gamma \times 2t}$
foreach $(i, j) \in [\gamma] \times [2t]$ do
  if $\boldsymbol{M}'[i, j] \neq 0$
    $\boldsymbol{D}'[i, j] := \mathsf{DummyIndex}\left(\boldsymbol{k}, \frac{\boldsymbol{K}'[i,j]}{\boldsymbol{M}'[i,j]}\right)$
return $\boldsymbol{D}'$
```

```
$\mathsf{Update}_2(d, k, m)$
foreach $i \in [\gamma]$ do
  $j := \mathsf{PRF}(s_2, (i, d))$
  $\boldsymbol{M}'[i, j] := \boldsymbol{M}'[i, j] - m$
  $\boldsymbol{K}'[i, j] := \boldsymbol{K}'[i, j] - k$
  if $\boldsymbol{M}'[i, j] \neq 0$
    $\boldsymbol{D}'[i, j] := \mathsf{DummyIndex}\left(\boldsymbol{k}, \frac{\boldsymbol{K}'[i,j]}{\boldsymbol{M}'[i,j]}\right)$
  else
    $\boldsymbol{D}'[i, j] := \perp$
```

Fig. 6. The second hybrid scheme works exactly as the first hybrid scheme, except that instead of using the wunder pseudorandom vector it uses the dummy sampler and the dummy index recovery to work with a uniformly random vector $\boldsymbol{k} \in ✄ K^n$.

```
$\mathsf{Compress}_3(\boldsymbol{m})$
$\boldsymbol{k} \leftarrow \mathsf{DummySample}(1^\lambda, n)$
$s_2 \leftarrow \{0,1\}^\lambda$
$\boldsymbol{M} := 0^{\gamma \times 2t}$
$\boldsymbol{K} := 0^{\gamma \times 2t}$
$\boldsymbol{C} := 0^{\gamma \times 2t}$
foreach $(i, d) \in [\gamma] \times [n]$ do
  $j := \mathsf{PRF}(s_2, (i, d))$
  $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] + m_d$
  $k := k_d$
  $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] + (k \cdot m_d)$
  if $m_d \neq 0$ do
    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] + 1$
return $(\boldsymbol{k}, s_2, \boldsymbol{M}, \boldsymbol{K}, \boldsymbol{C})$
```

```
$\mathsf{Decompress}_3((\boldsymbol{k}, s_2, \boldsymbol{M}, \boldsymbol{K}, \boldsymbol{C}))$
$S := \emptyset$
$\boldsymbol{M}' := \boldsymbol{M}$
$\boldsymbol{K}' := \boldsymbol{K}$
$\boldsymbol{D}' := \mathsf{Initialize}_2()$
while $\exists\, (i^*, j^*) \in [\gamma] \times [2t].\ \boldsymbol{C}[i^*, j^*] = 1$ do
  $(d, k, m) := (\boldsymbol{D}'[i^*, j^*], \boldsymbol{K}'[i^*, j^*], \boldsymbol{M}'[i^*, j^*])$
  $S := S \cup \{(d, m)\}$
  $\mathsf{Update}_3(d, k, m)$
return $S$
```

```
$\mathsf{Update}_3(d, k, m)$
foreach $i \in [\gamma]$ do
  $j := \mathsf{PRF}(s_2, (i, d))$
  $\boldsymbol{M}'[i, j] := \boldsymbol{M}'[i, j] - m$
  $\boldsymbol{K}'[i, j] := \boldsymbol{K}'[i, j] - k$
  $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] - 1$
  if $\boldsymbol{M}'[i, j] \neq 0$
    $\boldsymbol{D}'[i, j] := \mathsf{DummyIndex}\left(\boldsymbol{k}, \frac{\boldsymbol{K}'[i,j]}{\boldsymbol{M}'[i,j]}\right)$
  else
    $\boldsymbol{D}'[i, j] := \perp$
```

Fig. 7. The third hybrid scheme works exactly as the second hybrid scheme, except that it maintains a matrix counting how many non-zero messages are mapped to each individual cell and deciding which messages to peel based on these exact counts instead of relying on the matrix $\boldsymbol{K}$.

Claim 11. *It holds that*

$$\left|\begin{array}{l}\Pr[\mathsf{Decompress}_2(\mathsf{Compress}_2(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))] \\ -\Pr[\mathsf{Decompress}_3(\mathsf{Compress}_3(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))]\end{array}\right| \leq 2^{-\kappa}.$$

Proof. We first note two things

1. Whenever a correct element (m, d) is peeled, the resulting matrices $(\boldsymbol{C'}, \boldsymbol{M'}, \boldsymbol{K'}, \boldsymbol{D})$ are identical to the scenario where $m_d = \mathsf{Dec}(\mathsf{sk}, c_d) = 0$ and all other mesages are unchanged.
2. In the third hybrid scheme only correct elements are peeled.

The first observation follows because a correct peeling removes a message from the relevant cells by subtracting the corresponding values, which is equivalent to not adding them in the first place, which is exactly what happens if the message is zero. The second observation follows because we correctly keep track of the number of non-zero elements in each cell and only peel those, where a single non-zero element remains. By these observations, at any point during the execution of the decompression loop, there exists a vector $\boldsymbol{m'} \in \mathbb{F}_q^n$, such that for all (i, j)

$$\boldsymbol{K'}[i,j] := \begin{cases} d & \text{if } \frac{\sum_{\iota\in I_i} k_\iota m_\iota}{\sum_{\iota\in I_i} m_\iota} = k_d \\ \perp & \text{otherwise}\end{cases}$$

where $I_i = \{\iota \in [n] \mid \mathsf{PRF}(s_1, (i, \iota)) = j\}$.

We denote by $E_{r,i,j}$ the event that before the r-th iteration of the main loop of $\mathsf{Decompress}_4$, it holds that $C[i, j] > 1$ but $K[i, j] \neq \perp$. Note that $\mathsf{Decompress}_4$ and $\mathsf{Decompress}_3$ behave identically unless at least one of $E_{r,i,j}$ for $(r, i, j) \in [n] \times [\gamma] \times [2t]$ occurs.

Therefore by a union bound and Corollary 7

$$\left|\begin{array}{l}\Pr[\mathsf{Decompress}_3(\mathsf{sk},\mathsf{Compress}_3(\mathsf{pk},\boldsymbol{c})) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))] \\ -\Pr[\mathsf{Decompress}_4(\mathsf{Compress}_4(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))]\end{array}\right|$$

$$\leq \sum_{(r,i,j)\in[n]\times[\gamma]\times[2t]} \Pr[E_{r,i,j}] \leq \sum_{(r,i,j)\in[n]\times[\gamma]\times[2t]} \frac{n^2+n}{|K|} = \frac{(2t\gamma)(n^3+n^2)}{|K|} \leq \frac{(2t\gamma)(n^3+n^2)}{2^\kappa \cdot (2t\gamma)(n^3+n^2)} \leq 2^{-\kappa}$$

□

Claim 12. *It holds that*

$$\left|\begin{array}{l}\Pr[\mathsf{Decompress}_3(\mathsf{Compress}_3(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))] \\ =\Pr[\mathsf{Decompress}_4(\mathsf{Compress}_4(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))]\end{array}\right|.$$

Proof. The only difference between the two hybrids could occur, if when peeling a message from cell (i, j), the content of $\boldsymbol{D}[i, j]$ would differ between the two hybrids. However, this is not possible, since in hybrid three we have

$$\begin{aligned}\boldsymbol{D}[i,j] &= \mathsf{DummyIndex}(\boldsymbol{k}, \frac{\boldsymbol{K}[i,j]}{\boldsymbol{M}[i,j]}) \\ &= \mathsf{DummyIndex}(\boldsymbol{k}, \frac{k_d \cdot m_d}{m_d}) = \mathsf{DummyIndex}(\boldsymbol{k}, k_d) = d\end{aligned}$$

just as in hybrid four. □

The fifth hybrid is identical to the fourth hybrid except that the now unneccessary matrix $\boldsymbol{K}$ is removed. The following claim trivially follows from the fact that $\boldsymbol{K}$ is not used in either hybrids.

Claim 13. *It holds that*

$$\left|\begin{array}{l}\Pr[\mathsf{Decompress}_4(\mathsf{Compress}_4(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))] \\ =\Pr[\mathsf{Decompress}_5(\mathsf{Compress}_5(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk},\boldsymbol{c}))]\end{array}\right|.$$

```
$\mathsf{Compress}_4(\boldsymbol{m})$
$\boldsymbol{k} \leftarrow \mathsf{DummySample}(1^\lambda, n)$
$s_2 \leftarrow \{0,1\}^\lambda$
$\boldsymbol{M} := 0^{\gamma \times 2t}$
$\boldsymbol{K} := 0^{\gamma \times 2t}$
$\boldsymbol{C} := 0^{\gamma \times 2t}$
$\boldsymbol{D} := 0^{\gamma \times 2t}$
foreach $(i, d) \in [\gamma] \times [n]$ do
  $j := \mathsf{PRF}(s_2, (i, d))$
  $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] + m_d$
  $k := k_d$
  $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] + (k \cdot m_d)$
  if $m_d \neq 0$ do
    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] + 1$
    $\boldsymbol{D}[i, j] := \boldsymbol{D}[i, j] + d$
return $(\boldsymbol{k}, s_2, \boldsymbol{M}, \boldsymbol{K}, \boldsymbol{C}, \boldsymbol{D})$
```

```
$\mathsf{Decompress}_4((\boldsymbol{k}, s_2, \boldsymbol{M}, \boldsymbol{K}, \boldsymbol{C}, \boldsymbol{D}))$
$S := \emptyset$
$\boldsymbol{M}' := \boldsymbol{M}$
$\boldsymbol{K}' := \boldsymbol{K}$
$\boldsymbol{D}' := \boldsymbol{D}$
while $\exists\, (i^*, j^*) \in [\gamma] \times [2t].\ \boldsymbol{C}[i^*, j^*] = 1$ do
  $(d, k, m) := (\boldsymbol{D}'[i^*, j^*], \boldsymbol{K}'[i^*, j^*], \boldsymbol{M}'[i^*, j^*])$
  $S := S \cup \{(d, m)\}$
  $\mathsf{Update}_4(d, k, m)$
return $S$
```

```
$\mathsf{Update}_4(d, k, m)$
foreach $i \in [\gamma]$ do
  $j := \mathsf{PRF}(s_2, (i, d))$
  $\boldsymbol{M}'[i, j] := \boldsymbol{M}'[i, j] - m$
  $\boldsymbol{K}'[i, j] := \boldsymbol{K}'[i, j] - k$
  $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] - 1$
  $\boldsymbol{D}'[i, j] := \boldsymbol{D}'[i, j] - d$
```

Fig. 8. The fourth hybrid scheme works exactly as the third hybrid scheme, except that the matrix $\boldsymbol{D}$ which before contained the indices of the messages *if* it could be inferred from the matrix $\boldsymbol{K}$ is now maintained with a sum of the indices of all messages mapped to the cell. This means that whenever $\boldsymbol{C}[i, j] = 1$, $\boldsymbol{D}[i, j]$ contains the index of the single non-zero message mapped to cell (i, j).

```
$\mathsf{Compress}_6^{h(\cdot)}(\boldsymbol{m})$
$\boldsymbol{M} := 0^{\gamma \times 2t}$
$\boldsymbol{C} := 0^{\gamma \times 2t}$
$\boldsymbol{D} := 0^{\gamma \times 2t}$
foreach $(i, d) \in [\gamma] \times [n]$ do
  $j := h(i, d)$
  $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] + m_d$
  if $m_d \neq 0$ do
    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] + 1$
    $\boldsymbol{D}[i, j] := \boldsymbol{D}[i, j] + d$
return $(\boldsymbol{M}, \boldsymbol{C}, \boldsymbol{D})$
```

```
$\mathsf{Decompress}_6^{h(\cdot)}((\boldsymbol{M}, \boldsymbol{C}, \boldsymbol{D}))$
$S := \emptyset$
$\boldsymbol{M}' := \boldsymbol{M}$
$\boldsymbol{D}' := \boldsymbol{D}$
while $\exists\, (i^*, j^*) \in [\gamma] \times [2t].\ \boldsymbol{C}[i^*, j^*] = 1$ do
  $(d, m) := (\boldsymbol{D}'[i^*, j^*], \boldsymbol{M}'[i^*, j^*])$
  $S := S \cup \{(d, m)\}$
  $\mathsf{Update}_5(d, m)$
return $S$
```

```
$\mathsf{Update}_6(d, m)$
foreach $i \in [\gamma]$ do
  $j := h(i, d)$
  $\boldsymbol{M}'[i, j] := \boldsymbol{M}'[i, j] - m$
  $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] - 1$
  $\boldsymbol{D}'[i, j] := \boldsymbol{D}'[i, j] - d$
```

Fig. 9. The sixth hybrid scheme works exactly as the fifth hybrid scheme, except that instead of using a pseudorandom function to derive j from (i, d), it uses a truly random function h given as an oracle.

Claim 14. *If* PRF *is a secure pseudorandom function then it holds that*

$$\left|\begin{array}{l}\Pr[\mathsf{Decompress}_5(\mathsf{Compress}_5(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))] \\ - \Pr[\mathsf{Decompress}_6^{h(\cdot,\cdot)}(\mathsf{Compress}_6^{h(\cdot,\cdot)}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))]\end{array}\right| \le \mathsf{negl}(\lambda).$$

Proof. As the only difference between the two hybrids is the use of the function $\mathsf{PRF}(s_2, \cdot, \cdot)$ in the fifth hybrid and $h(\cdot, \cdot)$ in the sixth hybrid, the claims follows from a straightforward reduction that, given access to an oracle o, executes $\mathsf{Decompress}_6^{o(\cdot,\cdot)}(\mathsf{Compress}_6^{o(\cdot,\cdot)}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c})))$ and outputs 0 if the result equals $\mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))$ and 1 otherwise. □

Claim 15. *It holds for any vector of ciphertexts with hamming weight at most* t *that*

$$\Pr[\mathsf{Decompress}_6(\mathsf{Compress}_6(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))] \le \mathcal{O}(2^{-\kappa})$$

Proof. Denote $\boldsymbol{m} := \mathsf{Dec}(\mathsf{sk}, \boldsymbol{c})$, $S := \{(d, m') \mid m' = m_d\}$ and $S_{\neq 0} := \{(d, m') \in S \mid m' \neq 0\} = \mathsf{sparse}(\boldsymbol{m})$. Since $\boldsymbol{c}$ has Hamming weight at most t, we have that $|S_{\neq 0}| \le t$.

Comparing hybrid six with the definition of an IBLT in Figure 1, and keeping in mind Remark 1 we can observe, that what $\mathsf{Compress}_6$ actually outputs is simply an IBLT for pairs containing all elements of $S_{\neq 0}$ using hash functions $h(i, \cdot)$. Further, $\mathsf{Decompress}_6$ is in fact the same as List with Update_6 being identical to Peel. And since $|S_{\neq 0}| \le t$ and random functions are t-wise independent, it thus holds by Theorem 3 that

$$\begin{aligned}&\Pr[\mathsf{Decompress}_6(\mathsf{Compress}_6(\boldsymbol{m})) = \mathsf{sparse}(\boldsymbol{m})] \\ =&\Pr[\mathsf{List}(\mathsf{Insert}(B_0, S_{\neq 0})) = S_{\neq 0}] \ge 1 - \mathcal{O}(2^{-\gamma \log t}) = 1 - \mathcal{O}(2^{-\kappa})\end{aligned}$$ □

By combining all of the above claims and using the triangle inequality it follows that

$$\Pr[\mathsf{Decompress}(\mathsf{sk}, \mathsf{Compress}(\mathsf{pk}, \boldsymbol{c})) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c}))] \ge 1 - \mathcal{O}(2^{-\kappa}) - \mathsf{negl}(\lambda)$$

as claimed. □

References

AFS18. Adi Akavia, Dan Feldman, and Hayim Shaul. Secure search on encrypted data via multi-ring sketch. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 985–1001, Toronto, ON, Canada, October 15–19, 2018. ACM Press. doi:10.1145/3243734.3243810. 1

BDOP04. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-24676-3_30. 1

BT88. Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynominal interpolation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 301–309, Chicago, IL, USA, May 2–4, 1988. ACM Press. doi:10.1145/62212.62241. 2.2

CDG+21. Seung Geol Choi, Dana Dachman-Soled, S. Dov Gordon, Linsheng Liu, and Arkady Yerukhimovich. Compressed oblivious encoding for homomorphically encrypted search. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2277–2291, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press. doi:10.1145/3460120.3484792. 1, 1.1

CK20. Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 44–75, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-45721-1_3. 5.1

CKK16. Jung Hee Cheon, Miran Kim, and Myungsun Kim. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. *IEEE Transactions on Information Forensics and Security*, 11(1):188–199, January 2016. doi:10.1109/TIFS.2015.2483486. 1

CKL15. Jung Hee Cheon, Miran Kim, and Kristin E. Lauter. Homomorphic computation of edit distance. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *FC 2015 Workshops*, volume 8976 of *Lecture Notes in Computer Science*, pages 194–212, San Juan, Puerto Rico, January 30, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-48051-9_15. 1

CT65. James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, April 1965. 2.2

dP95. Gaspard de Prony. Essai expérimental et analytique sur les lois de la dilatabilité des fluides élastiques et sur celles de la force expansive de la vapeur de l'eau et de la vapeur de l'alcool à différentes températures. *Journal de l'École Polytechnique*, 1(22):24–76, 1795. 2.2

Gen09. Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009. 1

GM11. Michael T Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 792–799. IEEE Computer Society Press, September 28–30, 2011. doi:10.1109/Allerton.2011.6120248. 1.1, 2.3, 3

HG19. Qiao-Long Huang and Xiao-Shan Gao. Revisit sparse polynomial interpolation based on randomized kronecker substitution. In Matthew England, Wolfram Koepf, Timur M. Sadykov, Werner M. Seiler, and Evgenii V. Vorozhtsov, editors, *CASC 2019: 21st International Workshop on Computer Algebra in Scientific Computing*, volume 11661, pages 215–235, Moscow, Russia, August 26–30, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-26831-2_15. 2.2, 1

LLN15. Kristin E. Lauter, Adriana López-Alt, and Michael Naehrig. Private computation on encrypted genomic data. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014: 3rd International Conference on Cryptology and Information Security in Latin America*, volume 8895 of *Lecture Notes in Computer Science*, pages 3–27, Florianópolis, Brazil, September 17–19, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-16295-9_1. 1

LT22. Zeyu Liu and Eran Tromer. Oblivious message retrieval. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 753–783, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany. doi:10.1007/978-3-031-15802-5_26. 1, 1.1

RAD78. Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In Richard A. DeMillo, Richard J. Lipton, David P. Dobkin, and Anita K. Jones, editors, *Foundations of secure computation*, pages 169–179. Academic Press, 1978. 1

Sha71. Daniel Shanks. Class number, a theory of factorization, and genera. In Donald John Lewis, editor, *1969 Number Theory Institute*, volume 20 of *Proceedings of Symposia in Pure Mathematics*, pages 415–440. American Mathematical Society, 1971. 2.2

SWP00. Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55, Oakland, CA, USA, May 2000. IEEE Computer Society Press. doi:10.1109/SECPRI.2000.848445. 1

YSK$^+$13. Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. Secure pattern matching using somewhat homomorphic encryption. In Ari Juel and Bryan Parno, editors, *CCSW 2013: The ACM Cloud Computing Security Workshop*, pages 65–76, Berlin, Germany, November 8, 2013. ACM Press. doi:10.1145/2517488.2517497. 1